Methodology

for

ADV Requirements

that are

Explicitly Defined

for

Medium Robustness

Chapter 8

Development activity (ADV)

*Editor Note: **This provides the methodology for the explicit ADV requirements contained within the Medium Robustness PP Consistency Manual.***

*Editor Note: **This methodology is derived from an early version of proposed updated requirements; as such, the components were numbered with an approximation of where they might fall within each family (e.g. ADV_HLD.2). This differs with the identification of the components within the Medium Robustness definition, which identifies all of these components as being the first component within the family, and includes "_(EXP)" in the component label, such as ADV_HLD_(EXP).1. Although these labels differ, the contents of this methodology is identical; the labels herein were not updated, in order to preserve cross-referencing.)***

8.1 Evaluation of architectural design

8.1.1 Sub-activity ADV_ARC.1

8.1.1.1 Objectives

686 The objective of this sub-activity is to determine whether the architecture of the TSF supports the FPT_SEP and FPT_RVM components of domain separation and non-bypassability, respectively.

8.1.1.2 Application Notes

687 Evaluators should recognize that two distinct types of information should be present. The first is design information on how the system is constructed with respect to the domain separation mechanisms and non-bypassability architecture. The second is a justification for why the mechanisms described in the first type of in formation are sufficient. It is not acceptable to use the fact that the design information describes the domain separation mechanisms as a justification that domain separation is achieved.

688 Evaluators should also recognize that the design information for the non-bypassability aspect of this family is likely to be sparse when compared to that for the domain separation requirement. This is a consequence of FPT_RVM being concerned with interfaces that bypass the enforcement mechanisms. In most cases this is a consequence of the implementation, where if a programmer is writing an interface that accesses or manipulates an object, it is that programmer's responsibility to use interfaces that are part of the TSP enforcement mechanism for the object and not to try to "go around" those interfaces.

8.1.1.3 Input

689 The evaluation evidence for this sub-activity is:

a) the ST;

b) the functional specification;

c) the high-level design;

d) the low-level design; and

e) the implementation representation.

8.1.1.4 Evaluator actions

690 This sub-activity comprises two CC Part 3 evaluator action elements:

a) ADV_ARC.1.1E, and

b) ADV_ARC.1.2E.

8.1.1.4.1 Action ADV_ARC.1.1E

ADV_ARC.1.1C

CC Text: ***The presentation of the architectural design of the TSF shall be informal.***

ADV_ARC.1-1 The evaluator *shall examine* the architectural design to determine that it contains all necessary informal explanatory text.

691 If the entire architectural design is informal, this work unit is not applicable and is therefore considered to be satisfied.

692 Supporting narrative descriptions are necessary for those portions of the architectural design that are difficult to understand only from the semiformal or formal description (for example, to make clear the meaning of any formal notation).

ADV_ARC.1.2C

CC Text: ***The architectural design shall be internally consistent.***

ADV_ARC.1-2 The evaluator *shall examine* the presentation of the architectural design to determine that it is internally consistent.

693 The evaluator ensures that if identical functionality is described in more than one place in a document the descriptions contain no disparities. The evaluator also examines the design to ensure that the mechanisms described do not appear to

conflict, and to ensure that any potential conflicts are understood from the design description.

694 For further guidance on consistency analysis see Chapter 14.3.

ADV_ARC.1.3C

CC Text: ***The architectural design shall describe the design of the TSF self-protection mechanisms.***

ADV_ARC.1-3 The evaluator *shall examine* the architectural design to determine that the description of the self-protection mechanisms is presented at a level of detail similar to that present in the lowest level design decomposition.

695 The words "level of detail similar to that present in the lowest level design decomposition" are used to indicate that it is not required for the architectural description to contain detail one would expect to find in a low-level design (that is, implementation-level information) if the ST only contains FSP and HLD requirements. In performing this work unit the evaluator should first be familiar with the level of detail contained in all other design decompositions for the TSF.

696 If the lowest design decomposition is a functional specification, the evaluators should ensure that the self-protection mechanisms described cover those mechanisms whose effects are evident at the user interface. Such mechanisms include (but are not limited to) protection placed on the executable images of the TSF, and protection placed on containers (e.g., files) that hold data used by the TSF. The evaluators also ensure that the mechanisms that might be invoked through TSFI (e.g., state-transition mechanisms) are described.

697 If a high-level design is available, in addition to the information above the evaluators also ensure the description contains information on how architectural elements that contribute to TSF domain separation work. This would include (but not be limited to) information on how memory management is performed (including any "short-cuts" such as cache management and translation lookaside buffer management), processor (hardware) features such as a "ring" architecture and support for hardware page protection and context switching, and software constructs (e.g., a "switch table" for system call state transition) that are used to help enforce domain separation.

698 If low-level design is available, in addition to the information previously covered the architectural description should also contain information that is fairly close to the actual implementation of the functionality, and not so abstract that it is implementation independent. At this level, the description should contain (but not be limited to) information pertaining to coding conventions that would prevent TSF compromises (buffer overflows), and information on stack management for call and return operations. At this level the evaluator checks the descriptions of the mechanisms to ensure that the level of detail is such that there is little ambiguity between the description in the architectural design and the implementation representation. The evaluator should sample the implementation representation to ensure that the level of detail is appropriate.

ADV_ARC.1-4 The evaluator *shall examine* the architectural design to determine that the description of the self-protection mechanisms is complete.

699 Self-protection is typically achieved by a variety of mechanisms, ranging from physical and logical restrictions on access to the system; to hardware-based mechanisms such as "execution rings" and memory management functionality; to software-based mechanisms such as boundary checking of inputs on a trusted server. All such mechanisms should be described if appropriate to the lowest level of design decomposition.

700 The notion of "untrusted entities" is critical in performing this work unit. Untrusted entities are those that the TSF is to be protected from, and is specifically made with respect to which of the FPT_SEP components are included for the TOE. As it applies to all FPT_SEP components, the "untrusted entities" are those that are totally untrusted with respect to any TSP. One way to determine whether an entity is totally untrusted is that if a malicious person could write the code for the entity, and that entity still is not capability of affecting the TSF, then that entity can be untrusted without adversely affecting the TSP. For instance, although a sort routine running in the kernel of a Unix-like operating system may not, by itself, directly enforce any specific SFP, if it was written by a malicious developer it could compromise the TSF in any number of ways due to the fact that it is running in the kernel. Therefore, this program could not be untrusted.

701 For TOEs that include FPT_SEP.2, the architectural design discusses (in addition to the information mentioned in the previous paragraph) the protection mechanisms associated with the access control/information flow SFP named in the FPT_SEP requirement. Note that the "untrusted entities" for this discussion are everything but the entities implementing the named SFPs. In the previous example, the "untrusted entities" would now *include* the sort routine (assuming the sort routine was not part of the named SFPs). For TOEs that include FPT_SEP.3, the preceding applies except that the self-protection mechanisms for *all* access control/information flow SFPs are described in the high-level design.

702 Completeness of the description of the self-protection mechanisms implies that all of the mechanisms contributing to the domain separation functions are described, and described in a way that the evaluator can understand how they function to provide TSF self-protection. The evaluator should use knowledge gained from other evidence (functional specification, high-level design, low-level design, other parts of the architectural description, implementation representation, as included in the ST for the TOE) in determining if any functionality contributing to self-protection was described that is not present in the architectural design.

703 The evaluators should also think about ways in which the TSF might be compromised, and determine if there are mechanisms that prevent the compromises that they come up with. They could also look at the justification that the self-protection mechanisms meet the FPT_SEP requirement (provided by the vendor) to determine if there are aspects mentioned in that analysis that imply the existence of mechanisms that aren't described in the design.

ADV_ARC.1-5 The evaluator *shall examine* the architectural design to determine that the description of the self-protection mechanisms is accurate.

704 Accuracy of the description of the self-protection mechanisms is the property that the description faithfully describes what is implemented. The evaluator should use other evidence (functional specification, high-level design, low-level design, other parts of the architectural description, implementation representation, as included in the ST for the TOE) in determining whether there are discrepancies in any descriptions of the self-protection mechanisms. If ADV_IMP is included in the ST for the TOE, the evaluators should also sample the implementation representation to ensure that the descriptions are faithfully depicting the implementation. If an evaluator cannot understand how a certain self-protection mechanism works or could work in the system architecture, it may be the case that the description is not accurate.

ADV_ARC.1.4C

CC Text: ***The architectural design shall describe the design of the TSF in detail sufficient to determine that the security enforcing mechanisms cannot be bypassed.***

ADV_ARC.1-6 The evaluator *shall examine* the architectural design to determine that it presents an analysis that adequately describes how the TSF mechanisms cannot be bypassed.

Describing how the TSF mechanisms cannot be bypassed generally requires a systematic argument based on the TSP and the TSFI. The description of how the TSF works (contained in the design decomposition evidence, such as the functional specification, high-level and low-level design)—along with the information in the TSS—provides the background necessary for the evaluator to understand what resources are being protected and what security functions are being provided. The functional specification provides descriptions of the TSFI through which the resources/functions are accessed.

The evaluator should assess the description provided to ensure that, for each protected resource or function provided, an argument is made documenting that, for each TSFI, the interface either invokes an appropriate TSF mechanism or that the interface does not access the resource or require the provided function. Note that as with the TSF self-protection description, the level of this description should be commensurate with the lowest level of design decomposition evidence delivered for the TOE.

An example of a description follows. Suppose the TSF provides file protection. Further suppose that although the “traditional” system calls (TSFI) for open, read, and write invoke the file protection mechanism described in the high-level and low-level design, there exists a TSFI that allows access to a batch job facility (creating batch jobs, deleting jobs, modifying unrun jobs). The evaluator should be able to determine from the vendor-provided description that this TSFI invokes the same protection mechanisms as do the “traditional” interfaces. This could be done, for example, by referencing the appropriate sections of the high-level and low-level

design that discuss "how" the batch job facility TSFI achieves its security objectives.

Using this same example, suppose there is a TSFI whose sole purpose is to display the time of day. The evaluator should determine that the description adequately argues that this TSFI is not capable of manipulating any protected resources and should not invoke any security function.

For a final example using a security function rather than a protected resource, consider an ST that contains FCO_NRO.2, which requires that the TSF provides evidence of origination for information types specified in the ST. Suppose that the "information types" included all information that is sent by the TOE via e-mail. In this case the evaluator should examine the description to ensure that all TSFI that can be invoked to send e-mail perform the "evidence of origination generation" function are detailed. The description might point to user and administrative guidance to show all places where e-mail can originate (e.g., e-mail program, notification from scripts/batch jobs) and then how each of these places invokes the evidence generation function.

The evaluator should also ensure that the description is comprehensive, in that each interface is analyzed with respect to the entire TSP. This may require the evaluator to examine supporting information (functional specification, high-level design, low-level design, other parts of the architectural description, administrative and user guidance, and perhaps even the implementation representation, as provided for the TOE) to determine that the description has correctly capture all aspects of an interface. The evaluator should consider what portions of the TSP each TSFI might affect (from the description of the TSFI and its implementation in the supporting documentation), and then examine the description to determine whether it covers those aspects.

ADV_ARC.1.5C

CC Text: ***The architectural design shall justify that the design of the TSF achieves the self-protection function.***

ADV_ARC.1-7 The evaluator *shall check* the architectural design to determine that it contains a justification that is distinct from the design description of the self-protection mechanisms.

705 In addition to the actual presentation of the self-protection design, the evaluator ensures that the developer provides an argument (justification) for why the collection of self-protection mechanisms is sufficient to perform the TSF isolation function. It is not sufficient for the developer to claim this has been demonstrated solely by design presentation; additional prose is necessary.

ADV_ARC.1-8 The evaluator *shall examine* the architectural design to determine the justification provided is adequate.

706 The evaluators determine that the argument presented by the vendor addresses every portion of the TSF (e.g., kernel, administrative interface, remote user

interfaces, hardware). They also determine that the justification provides, for each portion of the TSF covered by a given mechanism, a rationale for why the identified mechanism(s) provide sufficient protection against untrusted users.

8.1.1.4.2 Action ADV_ARC.1.2E

CC Text: ***The evaluator shall analyze the architectural design and dependent documentation to determine that FPT_SEP and FPT_RVM are accurately implemented in the TSF.***

ADV_ARC.1-9 The evaluator *shall examine* the architectural design and all other design decomposition evidence supplied for the TOE to determine that FPT_SEP is achieved.

707 The vendor has presented their argument for why they think FPT_SEP is achieved in their implementation in the architectural design document. Using the information contained in that document, and information contained in all of the other evidence available for the TSF, the evaluator performs their own analysis to confirm that there is no way that an untrusted user can affect the TSF as required by the FPT_SEP requirement levied on the TOE. The evaluator ensures that information contained in the various design documents is consistent; inconsistent information or information "gaps" (a mechanism described in the high-level design, for instance, that is not mentioned in the low-level design) are usually flags for further investigation.

ADV_ARC.1-10 The evaluator *shall examine* the architectural design and all other design decomposition evidence supplied for the TOE to determine that FPT_RVM is achieved.

708 The vendor has presented their argument for why they think FPT_RVM is achieved in their implementation in the architectural design document. Using the information contained in that document, and information contained in all of the other evidence available for the TSF, the evaluator performs their own analysis to confirm that there is no way that an untrusted user can bypass the TSF mechanisms implemented to meet the SFRs contained in the ST.

709 The evaluator is expected to examine every interface to the extent that they can either confirm that the interface does not interface with a mechanisms implemented to satisfy an SFR, or confirm that interfaces that do perform functions that are covered by SFRs in the ST use the mechanisms described in the other design evidence.

710

8.2 Evaluation of functional specification

8.2.1 Sub-activity ADV_FSP.1

8.2.2 Sub-activity ADV_FSP.2

8.2.2.1 Objectives

711 The objective of this sub-activity is to determine whether the developer has completely identified all of the TSFI; whether the description of the security-enforcing TSFIs is complete and accurate; and whether the TSFI appear to implement the user-visible security functional requirements of the ST.

8.2.2.2 Application Note

712 An informal functional specification describes the interfaces to the TSF (the TSFI) in a manner that is not necessarily structured, as long as all of the requirements for this component are satisfied. Because of the dependency on ADV_HLD.1, the evaluator is expected to have identified the TSF prior to beginning work on this sub-activity. Without firm knowledge of what comprises the TSF, it is not possible to assess the completeness of the TSFI.

713 In performing the various work units included in this family, the evaluator is asked to make assessments of accuracy and completeness of several factors (the TSFI itself, as well as the individual components (parameters, effects, error messages, etc.) of the TSFI). In doing this analysis, the evaluator is expected to use the documentation provided for the evaluation in an iterative fashion. This includes the ST, user and administrative guidance, and the high-level design. It may also include the low-level design, the architectural design, architectural description, and the implementation representation of the TSF. The evaluator may read, for example, in the HLD how a certain function is implemented, but see no way to invoke that function from the interface. This might cause the evaluator to question the completeness of a particular TSFI description, or whether an interface has been left out of the functional specification altogether. Describing analysis activities of this sort in the ETR is a key method in providing rationale that the work units have been performed appropriately.

714 The focus of the work units generated by this component is three-fold. First, the entire TSFI must be described in enough detail so that a determination can be made about whether a particular TSFI has any security-enfocing aspects (that is, it relates directly to implementing one of the SFRs in the ST, with the exception of FPT_SEP or FPT_RVM) or whether the interface, or portions of an interface, and processing associated with that interface, must just maintain FPT_SEP and FPT_RVM (in which case the interface, or aspects of the interface, are considered security supporting).

715 Second, all security-enfocing components of an interface (exceptions, direct error messages, and effects) must be provided. It is important to note that generally the effects produced by an interface are not visible at that same interface; the effect still needs to be described if it is visible at any other TSFI. For instance, the effect

of an interface might include the production of an audit record. Although the invoker of that interface may not be able to tell that an audit record has been cut, the user of the audit tool would be able to tell.

716 Third, the evaluator must verify that all of the SFRs contained in the ST are completely and accurately allocated to the TSFI. This analysis is key in determining the soundness of the design. It should be noted that functional requirements that provide no interface but are architectural in nature (for example, FPT_SEP, FPT_RVM) are naturally not expressible or completely traceable in the functional specification.

8.2.2.3 Input

717 The evaluation evidence for this sub-activity that is required by the work-units is:

a) the functional specification;

b) the high-level design.

c) the low-level design;

d) the implementation representation;

e) the architectural design document;

f) the architectural description;

g) the ST;

h) the user guidance;

i) the administrator guidance.

8.2.2.4 Evaluator actions

718 This sub-activity comprises two CC Part 3 evaluator action elements:

a) ADV_FSP.2.1E;

b) ADV_FSP.2.2E.

8.2.2.4.1 Action ADV_FSP.2.1E

ADV_FSP.2.1C

CC Text : ***The functional specification shall completely represent the TSF.***

ADV_FSP.2-1 The evaluator *shall examine* the functional specification to determine that the TSF is fully represented.

719 The identification of the TSFI is a necessary prerequisite to all other activities in this sub-activity. The TSF must be identified (done as part of the ADV_HLD work units) in order to identify the TSFI. This activity can be done at a high level to ensure that no large groups of interfaces have been missed (network protocols, hardware interfaces, configuration files), or at a low level as the evaluation of the functional specification proceeds.

720 In order to assess the completeness of the TSF representation, the evaluator consults other evidence provided for the evaluation; this includes the TOE summary specification of the ST, the user guidance, the administrator guidance, and the high-level design; and may also include the low-level design, the architectural description, the architectural design, and the implementation representation. None of these should describe processing or effects that are absent from that described in the functional specification.

721 In order to identify the software interfaces to the TSF, the parts of the TOE that make up the TSF must be identified. This identification is formally a part of ADV_HLD analysis. In this analysis, a portion of the TOE is considered to be in the TSF under two conditions:

a) The software contributes to the satisfaction of security functionality specified by a functional requirement in the ST. This is typically all software that runs in a privileged state of the underlying hardware, as well as software that runs in unprivileged states that performs security functionality.

b) The software used by administrators in order to perform security management activities specified in the guidance documentation. These activities are a superset of those specified by any FMT_* functional requirements in the ST.

722 Identification of the TSFI is a complex undertaking. The TSF is providing services and resources, and so the TSFI are interfaces *to* the security services/resources the TSF is providing. This is especially relevant for TSFs that have dependencies on the IT environment, because not only is the TSF providing security services (and thus exposing TSFI), but it is also *using* services of the IT environment. While these are (using the general term) interfaces between the TSF and the IT environment, they are not TSFI. Nonetheless, it is vital to document their existence to integrators and consumers of the system, and thus documentation requirements for these interfaces are specified in ADV_CMP.

723 This concept (and concepts to be discussed in the following paragraphs) is illustrated in the following figure.

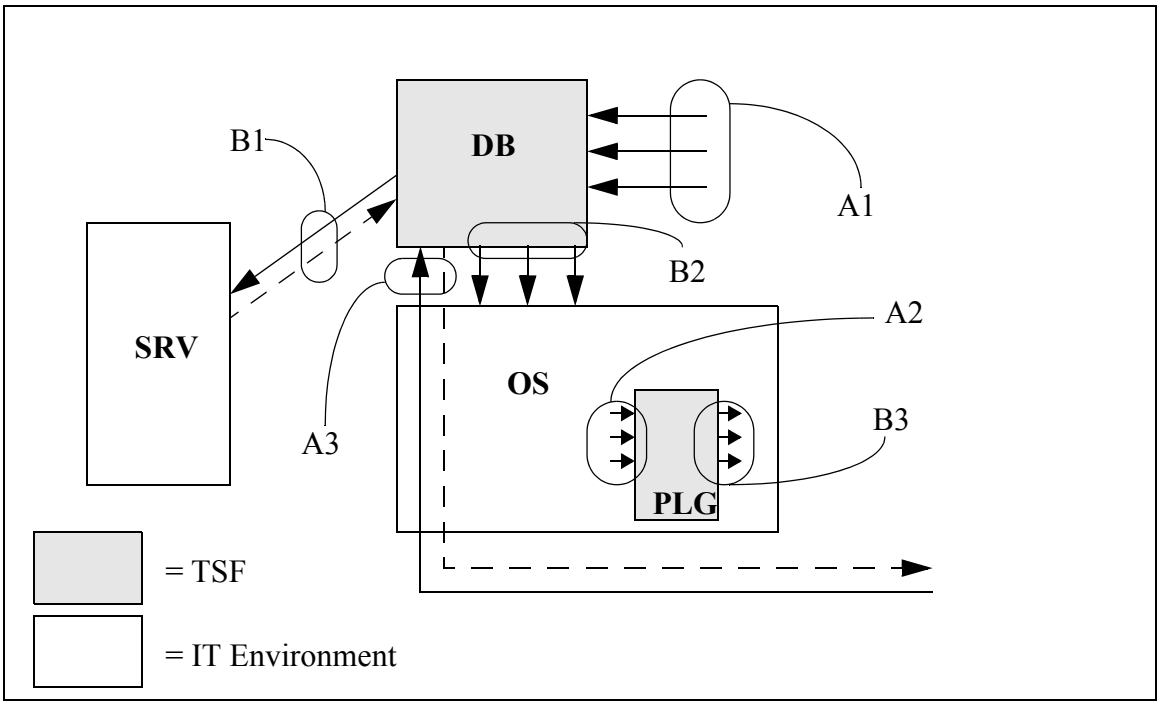


Figure 8.1 - Interfaces in a DBMS system

724 Figure 10.3 illustrates a TOE (a database management system) that has dependencies on the IT environment. The shaded boxes represent the TSF, while the unshaded boxes represent IT entities in the environment. The TSF comprises the database engine and management GUIs (represented by the box labelled "DB") and a kernel module that runs as part of the OS that performs some security function (represented by the box labelled "PLG"). The TSF kernel module has entry points defined by the OS specification that the OS will call to invoke some function (this could be a device driver, or an authentication module, etc.). The key is that this pluggable kernel module is providing security services specified by functional requirements in the ST. The IT environment consists of the operating system (represented by the box labelled "OS") itself, as well as an external server (labelled SRV). This external server, like the OS, provides a service that the TSF depends on, and thus needs to be in the IT environment. Interfaces in the figure are labelled Ax for TSFI, and Bx for interfaces to be documented in ADV_CMP. Each of these groups of interfaces is now discussed.

725 Interface group A1 represent the prototypical set of TSFI. These are interfaces used to directly access the database and its security functionality and resources.

726 Interface group A2 represent the TSFI that the OS invokes to obtain the functionality provided by the pluggable module. These are contrasted with interface group B3, which represent calls that the pluggable module makes to obtain services from the IT environment.

727 Interface group A3 represent TSFI that "pass through" the IT environment. In this case, the DBMS communicates over the network using a proprietary application-level protocol. While the IT environment is responsible for providing various supporting protocols (e.g., Ethernet, IP, TCP), the application layer protocol that is used to obtain services from the DBMS is a TSFI and must be documented as such. The dotted line indicates return values/services from the TSF over the network connection.

728 Non-TSFI interfaces pictured are labelled Bx. Interface group B1 is the most complex of these, because the architecture of the system and environmental assumptions and conditions will drive its analysis. In the first case, assume that, either through an environmental assumption or an IT environmental requirement, the network link between the DB and SRV is protected (it could be on a separate subnet, or it could be protected by a firewall such that only the DB could connect to the port on the SRV) such that only the DB has access to the SRV. In this case, the interface needs to be documented in the composition guidance (ADV_CMP) only, since untrusted users are unable to gain access.

729 However, consider the case where SRV is now just "somewhere on the network", and now the port that the DB opens up to communicate with the SRV is "exposed" to untrusted users. In this case, while the interface presented by the DB (the TSF) still only needs to be documented in ADV_CMP, additional considerations with respect to vulnerabilities may need to be documented as part of the AVA_VLA activity because of this exposure. In particular, since the TSF is receiving (and trusting) data from this external source, some consideration will have to be given (based on the desired assurance) to "man-in-the-middle" attacks.

730 In the course of performing its functions, the DB will make system calls down to the OS. This is represented by interface group B2. While these calls are not part of the TSFI, they are an interface that needs to be documented in the ADV_CMP.

731 Interface group B3, mentioned previously in connection with interface group A2, is similar to interface group B2 in that these are calls made by the TSF to the IT environment to perform services for the TSF.

732 Having discussed the interfaces in general, the types of TSFI are now discussed in more detail. This discussion categorizes the TSFI into the two categories mentioned previously: TSFI to software directly implementing the SFRs, and TSFI used by administrators.

733 TSFI in the first category are varied in their appearance in a TOE. Most commonly interfaces are thought of as those described in terms of Application Programming Interfaces (APIs), such as kernel calls in a Unix-like operating system. However, interfaces also may be described in terms of menu choices, check boxes, and edit boxes in a GUI; parameter files (the *.INI files and the registry for Microsoft Windows systems); and network communication protocols at all levels of the protocol stack.

734 TSFI in the second category are more complex. While there are three cases that need to be considered (discussed below), for all cases there is an "additional"

requirement that the functions that an administrator uses to perform their duties—as documented in administrative guidance—also are part of the TSFI and must be documented and shown to work correctly. The individual cases are as follows:

a) The administrative tool used is also accessible to untrusted users, and runs with some "privilege" itself. In this case the TSFI to be described are similar to those in the first category because the tool itself is privileged.

b) The administrative tool uses the privileges of the invoker to perform its tasks. In this case, the interfaces supporting the activities that the administrator is directed to do by the administrative guidance (AGD_ADM, including FMT_* actions) are part of the TSFI. Other interfaces supported by the tool that the administrator is directed not to use (and thus play no role in supporting the TSP), but that are accessible to non-administrators, are not part of the TSFI because there are no privileges associated with their use. Note that this case differs from the previous one in that the tool does not run with privilege, and therefore is not in and of itself interesting from a security point of view. Also note that if FPT_SEP is included in the ST, the executable image of such tools need to be protected so that an untrusted user cannot replace the tool with a "trojan" tool.

c) The administrative tool is only accessible to administrative users. In this case the TSFI are identified in the same manner as the previous case. Unlike the previous case, however, the evaluator ascertains that an untrusted user is unable to invoke the tool if FPT_SEP is included in the ST.

735 It is also important to note that some TOEs will have interfaces that one might consider part of the TSFI, but environmental factors remove them from consideration (an example is the case of interface group B1 discussed earlier). Most of these examples are for TOEs to which untrusted users have restricted access. For example, consider a firewall that untrusted users only have access to via the network interfaces, and further that the network interfaces available only support packet-passing (no remote administration, no firewall-provided services such as telnet). Further suppose that the firewall had a command-line interface that logged-in administrators could use to administer the system, or they could use a GUI-based tool that essentially translated the GUI-based checkboxes, textboxes, etc., into scripts that invoked the command-line utilities. Finally, suppose that the administrators were directed in the administrative guidance to use the GUI-based tool in administering the firewall. In this case, the command-line interface does not have to be documented because it is inaccessible to untrusted users, and because the administrators are instructed not use it.

736 The term "administrator" above is used in the sense of an entity that has complete trust with respect to all policies implemented by the TSF. There may be entities that are trusted with respect to some policies (e.g., audit) and not to others (e.g., a flow control policy). In these cases, even though the entity may be referred to as an "administrator", they need to be treated as untrusted users with respect to policies to which they have no administrative access. So, in the previous firewall example, if there was an auditor role that was allowed direct log-on to the firewall machine, the command-line interfaces not related to audit are now part of the TSFI, because they are accessible to a user that is not trusted with respect to the policies the

interfaces provide access to. The point is that such interfaces need to be addressed in the same manner as previously discussed.

737 Hardware interfaces exist as well. Functions provided by the BIOS of various devices may be visible through a "wrapper" interface such as the IOCTLs in a Unix operating system. If the TOE is or includes a hardware device (e.g., a network interface card), the bus interface signals, as well as the interface seen at the network port, must be considered "interfaces." Switches that can change the behaviour of the hardware are also part of the interface.

738 As indicated above, an interface exists at the TSF boundary if it can be used (by an administrator; untrusted user; or another TOE) to affect the behaviour of the TSF. The requirements in this family apply to all types of TSFI, not just APIs.

739 All TSFI are *security relevant*, but some interfaces (or aspects of interfaces) are more critical and require more analysis than other interfaces. If an interface plays a role in enforcing any security policy on the system (that is, if the effects of the interface can be traced to one of the SFRs levied on the TSF), then that interface is *security enforcing*. Such policies are not limited to the access control policies, but also refer to any functionality provided by one of the SFRs contained in the ST (with exceptions for FPT_SEP and FPT_RVM as detailed below). Note that it is possible that an interface may have various effects and exceptions, some of which may be security enforcing and some of which may not.

740 FPT_SEP and FPT_RVM are SFRs that require a different type of analysis from other SFRs. These requirements are architecturally related, and their implementation (or lack thereof) is not easily (or efficiently) testable at the TSFI. From a terminology standpoint, although implementation (and the associated analysis) of FPT_SEP and FPT_RVM is critical to the trustworthiness of the system, these two SFRs will not be considered as SFRs that are applicable when determining the set of security-enforcing TSFIs as defined in the previous paragraph.

741 Interfaces (or parts of an interface) that need only to function correctly in order for the security policies of the system to be preserved are termed *security supporting*. A security supporting interface typically plays a role in supporting the architectural requirements (FPT_SEP or FPT_RVM), meaning that as long as it can be shown that it does not allow the TSF to be compromised or bypassed no further analysis against SFRs is required. In order for an interface to be security supporting it must have *no* security enforcing aspects. In contrast, a security enforcing interface may have security supporting aspects (for example, the ability to set the system clock may be a security enforcing aspect of an interface, but if that same interface is used to display the system date that effect may only be security supporting).

ADV_FSP.2.2C

CC Text : ***The functional specification shall be internally consistent.***

ADV_FSP.2-2 The evaluator *shall examine* the functional specification to determine that it is internally consistent.

742 The evaluator validates the functional specification by ensuring that the descriptions of the interfaces making up the TSFI are consistent with similar interfaces. For example, if there are a set of interfaces that operate on files, and all but one of the interfaces takes a file handle, that presents a potential inconsistency that the evaluator should question.

743 Additionally, interfaces may produce effects visible at other interfaces. The description of the effects of interfaces should be consistent with the description of the functions of the interfaces at which those effects are visible. For instance, if an interface was used to set the access control permissions on an object, and the "access control permissions viewer" did not display some of the attributes that were settable (and described) for the "set" interface, it would present a potential inconsistency that the evaluator should question.

744 For further guidance on consistency analysis see Chapter 14.3.

ADV_FSP.2.3C

CC Text : ***The functional specification shall describe the external TSF interfaces (TSFI) using an informal style.***

ADV_FSP.2-3 The evaluator *shall examine* the functional specification to determine that it contains all necessary informal explanatory text.

745 If the entire functional specification is informal, this work unit is not applicable and is therefore considered to be satisfied.

746 Supporting narrative descriptions are necessary for those portions of the architectural design that are difficult to understand only from the semiformal or formal description (for example, to make clear the meaning of any formal notation).

ADV_FSP.1.4C

CC Text : ***The functional specification shall designate each external TSFI as security enforcing or security supporting.***

ADV_FSP.2-4 The evaluator *shall examine* the functional specification to determine that each TSFI is accurately identified as being "security enforcing" or "security supporting."

747 All TSFI are *security relevant*, but some interfaces (or aspects of interfaces) are more critical and require more analysis than other interfaces. If an interface plays a role in enforcing any security policy on the system, then that interface is *security enforcing*. Such policies are not limited to the access control policies, but also refer to any functionality provided by one of the SFRs contained in the ST. TSFI may also be security enforcing because they enforce a policy that may be different than their "advertised" functionality. For example, if an interface is capable of causing an audit record to be generated, it is security enforcing because it is helping to enforce the audit policy. Note that it is possible that interfaces may have various

effects and exceptions, some of which may be security enforcing and some of which may not. If there is at least one aspect of an interface that is security enforcing, the interface is designated as security enforcing.

748 Interfaces (or parts of an interface) that need only to function correctly in order for the security policies of the system to be preserved are termed *security supporting*. A security supporting interface typically supports FPT_SEP or FPT_RVM, meaning that as long as it can be shown that it does not allow the TSF to be compromised or bypassed no further analysis against SFRs is required. In order for an interface to be security supporting it must have *no* security enforcing aspects. In contrast, a security enforcing interface may have security supporting aspects (for example, the ability to set the system clock may be a security enforcing aspect of an interface, but if that same interface is used to display the system date that effect may only be security supporting).

749 The evaluators may wish to use the output of Action ADV_FSP.2.2E as input for this work unit, as any interface that aids in implementing an SFR (with the exception of FPT_SEP and FPT_RVM) is classified as security-enforcing. The evaluator should determine that all interfaces identified as security supporting have no security enforcing aspects by, for example, examining the high-level design, low-level design, architecture information, and implementation representation in order to verify that no TSP-enforcing functionality described by those decompositions is provided by a security supporting interface.

ADV_FSP.2.5C

CC Text : ***The functional specification shall describe the purpose and method of use for each external TSFI.***

ADV_FSP.2-5 The evaluator *shall examine* the functional specification to determine that it states the purpose of each TSFI.

750 The purpose of a TSFI is a general statement summarizing the functionality provided by the interface. It is not intended to be a complete statement of the effects of an interface, but rather a statement to help the reader understand in general what the interface is intended to be used for. The evaluator should not only determine that the purpose exists, but also that it accurately reflects the TSFI by taking into account other information about the interface, such as the description of effects and error messages; this can be done in association with work unit ADV_FSP.1-2, the internal consistency analysis.

ADV_FSP.2-6 The evaluator *shall examine* the functional specification to determine that the method of use for each TSFI is given.

751 The method of use for a TSFI summarizes how the interface is manipulated in order to cause the effect on the TSFI. The evaluator should be able to determine, from reading this material in the functional specification, how to use each interface. This does not necessarily mean that there needs to be a separate method of use for each TSFI, as it may be possible to describe in general how kernel calls are invoked, for instance, and then identify each interface using that general style.

Different types of interfaces will require different method of use specifications. APIs, network protocol interfaces, system configuration parameters, and hardware bus interfaces all have very different methods of use, and this should be taken into account by the developer when developing the functional specification, as well as by the evaluator evaluating the functional specification.

752 For administrative interfaces whose functionality is documented as being inaccessible to untrusted users, the evaluator ensures that the method of making the functions inaccessible is described in the functional specification. It should be noted that this inaccessibilty should be tested by the developer in their test suite.

753 The evaluator should not only determine that the set of method of use descriptions exist, but also that they accurately cover each TSFI; this can be done in association with work unit ADV_FSP.2-2, the internal consistency analysis.

ADV_FSP.2.6C

CC Text : ***The functional specification shall identify and describe all parameters associated with each external TSFI.***

ADV_FSP.2-7 The evaluator *shall examine* the presentation of the TSFI to determine that it completely identifies all parameters associated with every TSFI.

754 All of the parameters (not just those that the developer has termed "security-enforcing") are described for each TSFI. Parameters are explicit inputs or outputs to an interface that control the behaviour of that interface. For examples, parameters are the arguments supplied to an API; the various fields in packet for a given network protocol; the individual key values in the Windows Registry; the signals across a set of pins on a chip; etc.

755 In order to determine that *all* of the parameters are present in the TSFI, the evaluator should examine the rest of the interface description (effects, error messages, etc.) to determine if the effects of the parameter are accounted for in the description. The evaluator should also check other evidence provided for the evaluation (high-level design, low-level design, user and administrator guidance, implementation representation) to see if behaviour or additional parameters are described there but not in the functional specification.

ADV_FSP.2-8 The evaluator *shall examine* the presentation of the TSFI to determine that it completely and accurately describes all parameters associated with every TSFI.

756 Once all of the parameters have been identified, the evaluator needs to ensure that they are accurately described, and that the description of the parameters is complete. A parameter description tells what the parameter is in some meaningful way. For instance, the interface "foo(i)" could be described as having "parameter i which is an integer"; this is not an acceptable parameter description. A description such as "parameter i is an integer that indicates the number of users currently logged in to the system." is much more acceptable.

757 In order to determine that the description of the parameters is complete, the evaluator should examine the rest of the interface description (purpose, method of use, and effects, error messages, etc. if applicable) to determine if the effects of the parameter are accounted for in the description. The evaluator should also check other evidence provided (high-level design, low-level design, user and administrator guidance, implementation representation) to see if behaviour or additional parameters are described there but not in the functional specification.

ADV_FSP.2.7C

CC Text : ***For security enforcing external TSFIs, the functional specification shall describe the security-enforcing effects and security-enforcing exceptions.***

ADV_FSP.2-9 The evaluator *shall examine* the presentation of the TSFI to determine that it completely and accurately describes all security enforcing effects associated with security-enforcing TSFIs.

758 The evaluator checks to ensure that all of the security-enforcing effects are described. Effects of an interface describe what the interface does (as opposed to the high- and low-level design, which describe *how* the effect is provided by the TSF). The effects that need to be described in a functional specification are those that are visible at any external interface; not necessarily limited to the one being specified. For instance, the sole effect of an API call is not just the error code it returns. Also, depending on the parameters of an interface, there may be many different effects (for instance, an API might have the first parameter be a "subcommand", and the following parameters be specific to that subcommand. The IOCTL API in some Unix systems is an example of such an interface). Security-enforcing are those effects that play a direct role in implementing a SFR (other than FPT_SEP and FPT_RVM). Such effects may include the configuration of policy elements (e.g., setting the access control list of the file; setting a password) as well as the enforcement of a policy check (open file, write file, login, write audit record). Some security-enforcing effects may be more subtle; for instance, those associated with object reuse (FDP_RIP) if included in the ST.

759 In order to determine that the description of the effects of a TSFI are complete, the evaluator should review the rest of the interface description (parameter descriptions, error messages, etc.) to determine if the effects described are accounted for. If there are discrepancies, the evaluator will need to determine if this is due to a potential effect being non-security-enforcing, or if there is indeed an omission. The evaluator should also analyse other evidence provided for the evaluation (e.g., high-level design, low-level design, user and administrator guidance, implementation representation) to see if there is evidence of effects that are described there but not in the functional specification.

ADV_FSP.2-10 The evaluator *shall examine* the presentation of the TSFI to determine that it completely and accurately describes all security-enforcing exceptions associated with the security-enforcing TSFIs.

760 Exceptions refers to the processing associated with "special checks" that may be performed by an interface. An example would be an interface that has a certain set

of effects for all users except the superuser; this would be an exception to the normal effect of the interface. Because the nature of exceptions generally involves privilege, it should be the case that most, if not all, exceptions are security enforcing.

761 In systems that support a set of privileges that may be assigned to different users, exception processing is that processing associated with the behaviour of the interface in the presence of the privilege. For hardware TOEs, exception processing generally occurs when certain signals are received "out-of-band" and need to be handled in the current hardware context; in this case, if the signalling and associated processing is performing a function traced directly to a SFR, this would be a security-enforcing exception.

762 In order to determine that the description of exception processing of a TSFI is complete, the evaluator should rely primarily on other evidence provided for the evaluation (e.g., high-level design, low-level design, user and administrator guidance, implementation representation), especially evidence targeted at administrators or system programmers. The high- and low-level design might also describe exception processing that should be verified by the evaluator as being present in the interface description (as long as the effect of that processing is visible at the TSFI). The evaluator should once again ensure that any omissions due to an exception that is being categorized as not security enforcing (so it is security supporting) are supported by the evidence.

ADV_FSP.2.8C

CC Text : ***For security enforcing external TSFIs, the functional specification shall describe direct error messages resulting from security enforcing effects and exceptions.***

ADV_FSP.2-11 The evaluator *shall examine* the presentation of the TSFI to determine that it completely and accurately describes all direct errors associated with each security-enforcing TSFI.

763 Direct errors are defined as errors that are directly related to a TSFI; examples are API calls and parameter-checking for configuration files. Errors can take many forms, depending on the interface being described. For an API, the interface itself may return an error code or an error message; set a global error condition, or set a certain parameter with an error code. For a configuration file, an incorrectly configured parameter may cause an error message to be written to a log file. For a hardware PCI card, and error condition may raise a signal on the bus, or trigger an exception condition to the CPU.

764 The direct errors that need to be analyzed are limited to those that result from security-enforcing processing associated with a security-enforcing TSFI. Since TSFI can have both security-enforcing and security-supporting effects, processing that implements a security-supporting effect may trigger a direct error. Such errors do not need to be reported in the functional specification.

765 The evaluator determines that, for each security-enforcing TSFI, the exact set of error messages that can be returned on invoking security-enforcing aspects of that

interface can be determined. For this category of errors, the evaluator reviews the evidence provided for the interface to determine if the set of direct errors seems complete. They cross-check this information with other evidence provided for the evaluation (high-level design, low-level design, user and administrator guidance, implementation representation) to ensure that there are no direct errors steming from security-enforcing processing mentioned that are not included in the functional specification.

ADV_FSP.2-12 The evaluator *shall examine* the presentation of the TSFI to determine that it completely and accurately describes the meaning of all direct errors associated with each security-enforcing TSFI.

766 In order to determine accuracy, the evaluator must be able to understand meaning of the error. For example, if an interface returns a numeric code of 0, 1, or 2, the evaluator would not be able to understand the error if the functional specification only listed: "possible errors resulting from security-enforcing processing for the *foo()* interface are 0, 1, or 2." Instead the evaluator checks to ensure that the errors are described such as: "possible errors resulting from security-enforcing processing for the *foo()* interface are 0 (processing successful), 1 (file not found), or 2 (incorrect filename specification)."

767 In order to determine that the description of the errors related to a security-enforcing TSFI are complete, the evaluator should examine the rest of the interface description (parameter descriptions, security-enforcing effects, security-enforcing exception processing, etc.) to determine if potential error conditions that might be caused by using such an interface are accounted for. The evaluator should also check other evidence provided for the evaluation (e.g., high-level design, low-level design, user and administrator guidance, implementation representation) to see if error processing related to security-enforcing functionality is described there but is not described in the functional specification.

8.2.2.0.1 Action ADV_FSP.2.2E

CC Text: ***The evaluator shall determine that the functional specification is an accurate and complete instantiation of all user-visible TOE security functional requirements.***

ADV_FSP.2-13 The evaluator *shall examine* the functional specification to determine that it is a complete instantiation of the user-visible TOE security functional requirements.

768 To ensure that all ST security functional requirements are covered by the functional specification, as well as the test coverage analysis, the evaluator may construct a map between the TOE security functional requirements and the TSFI. Note that this map may have to be at a level of detail "below" the component or even element level of the requirements, because of operations (assignments, refinements, selections) performed on the functional requirement by the ST author.

769 For example, the FDP_ACF.1 component contains four elements, each of which has complex assignments. If the ST contained, for instance, 10 rules in the FDP_ACF.1.2 assignment, and these 10 rules were "covered" by 3 different TSFI, it would be inadequate for the evaluators to "map" FDP_ACF.1.2 to TSFI A, B,

and C and claim they had completed the work unit. Instead, the evaluators would map FDP_ACF.1.2 (rule 1) to TSFI A; FDP_ACF.1.2 (rule 2) to TSFI B; etc. It might also be the case that the interface is a "wrapper" interface (e.g., IOCTL), in which case the mapping would need to be specific to certain set of parameters for a given interface.

770 "user-visible" should be interpreted such that for requirements that have little or no manifestation at the user interface (e.g., FPT_SEP, FPT_RVM) it is not expected that the evaluator completely map those requirements to the TSFI. The analysis for those requirements will be performed in the analysis for the architectural design (ADV_ARC). It is also important to note that since the parameters, effects, exceptions, and direct error messages associated with security-enforcing aspects of TSFIs must be fully specified, the evaluator should be able to determine if all aspects of an SFR appear to be implemented at the interface level.

ADV_FSP.2-14 The evaluator *shall examine* the functional specification to determine that it is an accurate instantiation of the user-visible TOE security functional requirements.

771 For each functional requirement in the ST that results in user-visible effects, the information in the associated TSFI for that requirement specifies the required functionality described by the requirement. For example, if the ST contains a requirement for access control lists, and the only TSFI that map to that requirement specify functionality for Unix-style protection bits, then the functionality specification is not accurate with respect to the requirements.

772 "user-visible" should be interpreted such that for requirements that have little or no manifestation at the user interface (e.g., FPT_SEP, FPT_RVM) it is not expected that the evaluator completely analyze those requirements with respect to the TSFI. The analysis for those requirements will be performed in the analysis for the architectural design (ADV_ARC).

8.3 Evaluation of high-level design

8.3.1 Sub-activity ADV_HLD.1

8.3.2 Sub-activity ADV_HLD.2

8.3.2.1 Objectives

773 The objective of this sub-activity is to determine whether the high-level design provides a description of the TOE in terms of major structural units (i.e. subsystems) sufficient to determine the TSF boundary, provides a description of the high-level data flows between the structural units of the TSF, and appears to be a correct design decomposition of the functional specification, focusing on the security-enfocing aspects of the high-level design.

8.3.2.2 Application Notes

774 The architectural unit of interest in the high-level design is the subsystem. A subsystem description should provide information on *how* the security functionality is being provided (at a high level), and discuss general data and control flows between the subsystems of the TSF. The description should also provide sufficient information for the evaluators to determine if a subsystem of the TOE should or should not be included as part of the TSF; the level of detail required for a non-TSF subsystem is likely to be much less than that for a TSF subsystem.

775 For subsystems comprising the TSF, those that implement security-enforcing TSFI contain security-enforcing aspects that need to be discussed in the subsystem description. “Security enforcing” is used to to describe subsystems that implement at least one security-enforcing TSFI. The security-enforcing aspects of a subsystems are those that directly contribute to implementing an SFR other than FPT_SEP and FPT_RVM.

8.3.2.3 Input

776 The evaluation evidence for this sub-activity is:

a) the ST;

b) the high-level design;

c) the functional specification;

d) the low-level design;

e) the architectural design;

f) the composition information;

g) the architectural description; and

h) the implementation representation.

8.3.2.4 Evaluator actions

777 This sub-activity comprises two CC Part 3 evaluator action elements:

a) ADV_HLD.2.1E;

b) ADV_HLD.2.2.E.

8.3.2.4.1 Action ADV_HLD.2.1E

ADV_HLD.2.1C

CC Text: ***The high-level design shall describe the structure of the TOE in terms of subsystems.***

ADV_HLD.2-1 The evaluator *shall examine* the high-level design to determine that the structure of the TSF is described in terms of subsystems.

778 With respect to the high-level design, the term *subsystem* refers to large, related units (such as memory-management, file-management, process-management). Breaking a design into the basic functional areas aids in the understanding of the design.

779 The primary purpose for examining the high-level design is to aid the evaluator's understanding of the TOE. The developer's choice of subsystem definition, and of the grouping of TSFs within each subsystem, are an important aspect of making the high-level design useful in understanding the TOE's intended operation. As part of this work unit, the evaluator should make an assessment as to the appropriateness of the number of subsystems presented by the developer, and also of the choice of grouping of functions within subsystems. The evaluator should ensure that the decomposition of the TSF into subsystems is sufficient for the evaluator to gain a high-level understanding of how the functionality of the TSF is provided.

780 The subsystems used to describe the high-level design need not be called "subsystems", but should represent a similar level of decomposition. For example, the design may be decomposed using "layers" or "managers".

ADV_HLD.2.2C

CC Text: ***The high-level design shall be internally consistent.***

ADV_HLD.2-2 The evaluator *shall examine* the presentation of the high-level design to determine that it is internally consistent.

781 The evaluator ensures that if identical functionality is described in more than one place in a document the descriptions contain no disparities. The evaluator also

examines control or data flow descriptions for the same type of data flow (access control, memory management) to ensure they are described in a consistent manner.

782 For further guidance on consistency analysis see Chapter 14.3.

ADV_HLD.2.3C

CC Text: ***The high-level design shall describe the subsystems using an informal style.***

ADV_HLD.2-3 The evaluator *shall examine* the high-level design to determine that it contains all necessary informal explanatory text.

783 If the entire high-level design is informal, this work unit is not applicable and is therefore considered to be satisfied.

784 Supporting narrative descriptions are necessary for those portions of the high-level design that are difficult to understand only from the semiformal or formal description (for example, to make clear the meaning of any formal notation).

ADV_HLD.2.4C

CC Text: ***The high-level design shall describe the design of the TOE in sufficient detail to determine what subsystems of the TOE are part of the TSF.***

ADV_HLD.2-4 The evaluator *shall examine* the high-level design to determine that it describes the IT portions of the TOE in sufficient detail to determine whether a component of the TOE is part of the TSF.

785 One of the most important activities in designing and evaluating a trusted product is determining the "security perimeter"; for the purposes of this discussion we will refer to this as the TSF.

786 The IT portions of the TOE are defined to be that software and hardware that will be available (installed) once the system is configured according to the ADO_IGS guidance. For instance, if an operating system ships with a database product, but the IGS guidance tells the administrator not to install the database product, then this work unit does not apply to that database product. Conversely, if the database product gets installed by default when the ADO_IGS guidance is followed, even if the administrative guidance and user guidance does not mention this product, it still must be addressed with respect to this work unit.

787 Once the set of software and hardware has been determined according to the guidance above, the evaluators should assess the high-level design to ensure that every component "available" has a description enabling them to make a determination as to whether a given component is relevant to the security functionality of the system or not. The granularity of "component" is left to the author of the high-level design, as long as the evaluator can make the required determination. For components that are not security relevant, there should be a short description of the component and a very brief argument for why it is not security relevant. In making a determination of whether a subsystem should be

considered part of the TSF, the evaluator should consider whether it plays a role in implementing any SFR listed in the ST, including FPT_SEP and FPT_RVM. It is generally fairly straightforward to determine if a subsystem implements a mechanism such as I&A or audit. For FPT_SEP and FPT_RVM, a test the evaluators can use is to determine the impact if a malicious developer was allowed to make any modification they wished to the subsystem in question. If the malicious developer could not affect the correct enforcement of any SFP on the system, then the subsystem by definition is not part of the TSF. However, if a malicious developer could write a piece of code for the subsystem that would affect the security policies of the system, then that subsystem is likely part of the TSF. This is the reason that all code that runs in the most privileged hardware mode is generally part of the TSF, even though it may not directly implement any SFR (apart from FPT_SEP and FPT_RVM).

788 If the evaluator cannot make the determination of security relevence from the description given, the developer will need to provide additional detail. No detailed discussion or description needs to be given for components that are security relevant, because they will be described in detail when they are broken into subsystems of the TSF.

789 The TSF comprises all the parts of the TOE that have to be relied upon for enforcement of the TSP. Because the TSF includes both functions that directly enforce the TSP, and also those functions that, while not directly enforcing the TSP, contribute to the enforcement of the TSP in a more indirect manner, all TSP-enforcing subsystems are contained in the TSF. Subsystems that play no role in TSP enforcement are not part of the TSF. An entire subsystem is part of the TSF if any portion of it is. This determination of the TSF boundary is a critical step in the evaluation process, and should be performed by the evaluators early in the evaluation activity.

ADV_HLD.2.5C

CC Text: ***The high-level design shall identify all subsystems in the TSF, and designate them as either security enforcing or security supporting.***

ADV_HLD.2-5 The evaluator *shall examine* the high-level design to determine that all subsystems are identified and accurately designated as either security enforcing or security supporting.

790 Having identified the security boundary, the TSF is then described in terms of subsystems in the high-level design. The evaluator should ensure that the description of the TSF is in terms of subsystems; that is, relatively large portions of the TSF that provides a broad function or plays a broad role in the design of the TSF. Identification of a subsystem just means that each subsystem of the TSF contains a unique tag so that it can be referenced.

791 “Security enforcing” is used to to describe subsystems that implement at least one security-enforcing TSFI. Since a TSF subsystem must be either security enforcing or security supporting, the security-supporting subsystems are merely those that are not security enforcing.

792 In determining whether the designation of a subsystem as security enforcing or security supporting is accurate, the evaluator should examine the TSFI that the subsystem supports (if not evident from the high-level design itself, this information should be contained in the ADV_RCR mapping provided by the developer). If any of the TSFI supported by a subsystem are security enforcing, then that subsystem should be designated as security enforcing in the high-level design.

ADV_HLD.2.6C

CC Text: ***The high-level design shall describe the structure of the security-enforcing subsystems***

ADV_HLD.2-6 The evaluator *shall examine* the high-level design to determine that the security-enforcing subsystems of the TSF are described at an architectural level.

793 The scope of the high-level design description for the subsystems is that which is necessary to discuss how the security functionality presented by the TSFI is realized at a design level. The level of the discussion should be such that major data and control elements are defined, and the reader can obtain a conceptual view of *how* the system is designed to provide the functionality visible at the external interface (that is, the TSFI). In other words, the functional specification (through the TSFI) indicates *what* the TSF provides; the high-level design should describe *how* it is provided.

The evaluator should determine from the description of the security-enforcing aspects of each subsystem that the description indicates how the functionality is being provide. The evaluator should find unacceptable assertions of capability in the high-level design unless the assertions are backed by a description of why the assertion is true.

ADV_HLD.2.7C

CC Text: ***For security-enforcing subsystems, the high-level design shall describe the design of the security-enforcing behavior.***

ADV_HLD.2-7 The evaluator *shall examine* the high-level design to determine that each subsystem designated as "security enforcing" describes how it provides its security-enforcing functionality.

794 The specific contents of the high-level design will depend on the developing organization's policies in terms of how much detail is included. While implementation detail (for example, actual programming language data structures, pseudo-code, etc.) is not required, it may aid discussion of a subsystem's purpose and method of operation. The level of detail should minimally be that which would be given to a programmer to write the low-level design such that, with perhaps a minimal amount of interaction with other programmers for the same TOE, the low-level design would be realizable with a minimum of conflicts between the subsystems. At this level of abstraction, it is appropriate to describe logical groupings of data (for instance, the notions of inodes and u-areas for a Unix

system; an object-model for a system implemented in C++) and their function with respect to the subsystems without having to describe the actual data structures that implement the logical groupings of data.

795 An example of adequate detail in a high-level design description (using the 4.4 BSD operating system as an example) can be found in *The Design and Implementation of the 4.4 BSD Operation System* by M.K. McKusick, K. Bostic, M. Karels, and J. Quarterman (Addison-Wesley, 1996). For instance, Chapter 6 presents "The Structure of Processes" generally in a manner consistent with a high-level design, although there is some amount of lower-level detail present for purposes of illustration. Note that this reference is given with respect to the level of detail required; only the security-enforcing aspects of a subsystem need to be described in this manner.

ADV_HLD.2.8C

CC Text: ***For security-enforcing subsystems, the high-level design shall summarize any non-security-enforcing behavior.***

ADV_HLD.2-8 The evaluator *shall examine* the high-level design to determine that each subsystem designated as "security enforcing" summarizes its security supporting aspects.

796 For security-enforcing subsystems, there may be two types of architectural detail provided. The first type is detail provided to describe the design for security-enforcing TSFI. The focus of this type of presentation is on how the security-enforcing functionality is being provided by the TSF, and its analysis is covered by ADV_HLD.2-6 and ADV_HLD.2-7. The second type of detail is provided for the security-supporting aspects of subsystems designated as security enforcing. The focus of this type of detail is on how the subsystem design supports the FPT_SEP and FPT_RVM requirements. This is a different emphasis in that it is not discussing the design of a particular mechanism, but rather is a description of the design supporting the architectural goals of self-protection and non-bypassability. The high-level design may reference the architectural design (ADV_ARC) for the above details, since the details of the TSF's architecture concerning the FPT_SEP and FPT_RVM requirements are contained in the architectural design document. Note that the assessment of the adequacy of the description of the security-supporting aspects of the subsystems is evaluated while performing the ADV_ARC work units; the results of that analysis may be helpful in completing the analysis required for this work unit.

ADV_HLD.2.9C

CC Text: ***The high-level design shall summarize the behavior for security-supporting subsystems.***

ADV_HLD.2-9 The evaluator *shall examine* the high-level design to determine that each subsystem designated as "security supporting" summarizes its security-supporting aspects.

797 A security-supporting subsystem is one that provides no functionality associated with a security-enforcing TSFI. In other words, the only policies a security-supporting subsystem supports are TSF self-protection and non-bypassability of the TSF. The evaluator examines the subystem description to ensure that it provides sufficient detail so that the evaluator is convinced that the design of the subsystem will not impact the integrity of the TSF. This detail should not solely consist of an assertion that it has no impact, but should provide a design summary sufficient to be convincing that it has no impact. It may be the case that such a summary will not require a description of the object model used, data groupings, or other types of detail found in the subsystem descriptions for security-enforcing subsystems. The high-level design may reference the architectural design (ADV_ARC) for the above details, since the details of the TSF's architecture concerning the FPT_SEP and FPT_RVM requirements are contained in the architectural design document. Note that the assessment of the adequacy of the description of the security-supporting aspects of the subsystems is evaluated while performing the ADV_ARC work units; the results of that analysis may be helpful in completing the analysis required for this work unit.

ADV_HLD.2.11C

CC Text: ***The high-level design shall summarize all other interactions between subsystems of the TSF.***

ADV_HLD.2-10 The evaluator *shall examine* the high-level design to determine that it summarizes the interactions between the security-enforcing and security-supporting subsystems of the TSF.

In order to provide additional assurance that subsystems characterized as security supporting are correctly categorized, the evaluator analyzes the descriptions of interactions between security-supporting and security-enforcing modules. Here, "interaction" can be abstracted to the subsystem level. For instance, if security-supporting subsystems used the security-enforcing (because of FDP_RIP, for instance) subsystem "Memory Manager", then it would be sufficient to indicate that "Security-supporting subsystem *foo* interactions with the memory manger to obtain buffers for *whatever* and to obtain *other memory objects*." It is not necessary nor required to list the details of each interaction (or "call") between the subsystems.

The degree of detail should be sufficient so that the evaluator can conclude that the security-supporting designation is correct. This might entail providing a brief description of the purpose of the interaction, but it does not necessarily need to be the same level of detail as the descriptions for interactions between security-enforcing subsystems.

ADV_HLD.2-11 The evaluator *shall examine* the high-level design to determine that the description of the interactions between the security-supporting subsystems and security-enforcing subsystems of the TSF is complete.

798 "Complete" in the context of this work unit means that for all security-supporting subsystems, if a security-supporting subsystem interacts with a security-enfocing subsystem, that interaction is described. Here, "interaction" can be abstracted to

the subsystem level. For instance, if security-supporting subsystems used the security-enforcing (because of FDP_RIP, for instance) subsystem "Memory Manager", then it would be sufficient to indicate that "Security-supporting subsystem *foo* interactions with the memory manger to obtain buffers for *whatever* and to obtain *other memory objects*." It is not necessary nor required to list the details of each interaction (or "call") between the subsystems.

799 The evaluator should use their understanding of how the security-enforcing functionality is being provided, as well as the architecture of the TSF, to determine whether the set of interactions described in the high-level design appears to be complete. The evaluator should also examine the other evidence provided for the TOE (functional specification, low-level design, architectural description, architectural design, implementation representation) to determine that subystem interactions that might occur due to interfaces/functionality described in those documents is captured.

ADV_HLD.2.12C

CC Text: ***The high-level design shall describe any interactions between the security-enforcing subsystems of the TSF.***

ADV_HLD.2-12 The evaluator *shall examine* the high-level design to determine that it describes the interactions between the security-enforcing subsystems of the TSF.

800 In the high-level design, interactions between subsystems that have an impact on how the functionality described by the TSFI is implemented need to be specified to aid the reader in understanding the overall design of the system. These interactions do not need to be characterized at the implementation level (e.g., parameters passed from one routine in a subsystem to a routine in a different subsystem; global variables; hardware signals (e.g., interrupts) from a hardware subsystem to an interrupt-handling subsystem), but the data elements identified for a particular subsystems that are going to be used by another subsystems should be covered in this discussion if there is any security-relevant behavior. Any control relationships between subsystems (e.g., a subsystem responsible for configuring a rulebase for a firewall system and the subsystem that actually implements these rules) should also be described if they impact the security being provided by the TSF.

ADV_HLD.2-13 The evaluator *shall examine* the high-level design to determine that the description of the interactions between the security-enforcing subsystems of the TSF is complete.

801 "Complete" in the context of this work unit means that if any two security-enforcing subsystems interact, their interaction is described. The evaluator should use their understanding of how the security-enforcing functionality is being provided to determine whether the set of interactions described in the high-level design appears to be complete. The evaluator should also examine the other evidence provided for the TOE (functional specification, low-level design, architectural description, architectural design, implementation representation) to

determine that subystem interactions that might occur due to interfaces/ functionality described in those documents is captured.

8.3.2.4.2 Action ADV_HLD.2.2E

CC Text: ***The evaluator shall determine that the high-level design is an accurate and complete instantiation of all user-visible TOE security functional requirements with the exception of FPT_SEP and FPT_RVM.***

ADV_HLD.2-14 The evaluator *shall examine* the high-level design to determine that it is an accurate instantiation of the user-visible TOE security functional requirements with the exception of FPT_SEP and FPT_RVM.

802 The evaluator ensures that each security requirement listed in the TOE security functional requirements section of the ST has a corresponding design description in the high-level design that accurately details how the TSF is designed to meet that requirement. As an example, if the ST requirements specified a role-based access control mechanism and the high-level design described access control based on UNIX-style protection bits, the high-level design would not be an accurate instantiation of those access control requirements.

803 "user-visible" should be interpreted such that for requirements that have little or no manifestation at the user interface (e.g., FPT_SEP, FPT_RVM) it is not expected that the evaluator completely map those requirements to the TSFI. The analysis for those requirements will be performed in the analysis for the architectural design (ADV_ARC).

ADV_HLD.2-15 The evaluator *shall examine* the high-level design to determine that it is a complete instantiation of the user-visible TOE security functional requirements with the exception of FPT_SEP and FPT_RVM.

804 To ensure that all ST security functional requirements are covered by the high-level design, the evaluator may construct a map between the TOE security functional requirements and the high-level design. Note that this map may have to be at a level of detail "below" the component or even element level of the requirements, because of operations (assignments, refinements, selections) performed on the functional requirement by the ST author.

805 For example, the FDP_ACF.1 component contains four elements, each of which has complex assignments. If the ST contained, for instance, 10 rules in the FDP_ACF.1.2 assignment, and these 10 rules were implemented in specific places in two subsystems, it would be inadequate for the evaluators to "map" FDP_ACF.1.2 to subsystems A and B and claim they had completed the work unit. Instead, the evaluators would map FDP_ACF.1.2 (rule 1) to subsystem A (section a); FDP_ACF.1.2 (rule 2) to subsystem B (section x); etc.

806 "user-visible" should be interpreted such that for requirements that have little or no manifestation at the user interface (e.g., FPT_SEP, FPT_RVM) it is not expected that the evaluator completely map those requirements to the TSFI. The analysis for

those requirements will be performed in the analysis for the architectural design (ADV_ARC).

8.4 Evaluation of TSF internals

8.4.1 Subactivity ADV_INT.1

8.4.1.1 Objectives

807 The objective of this sub-activity is to determine whether the TSF is designed and structured in a modular fashion that ensures the TSF implementation is not overly complex and facilitates understanding by the developer and the evaluator.

8.4.1.2 Application notes

808 The role of the architectural description is to provide evidence of modularity of the TSF and to justify the complexity of the design and implementation. The focus of the architectural description is on the appropriateness of inclusion of functions in a module, and on the interaction between modules. The low-level design, on the other hand, describes the design of the modules of the TSF and how those modules work to satisfy the SFRs.

809 The modules identified in the architectural description are the same as the modules identified in the low-level design. Some of the information from the low-level design may be applicable in meeting the required content for the architectural description, and can be included by reference.

810 For portions of the TSF implemented in software, a module consists of one or more source code files that cannot be decomposed into smaller compilable units.

811 This explicit component levies stricter modularity constraints on the SFP-enforcing modules than on the non-SFP-enforcing modules. The SFP modules are those modules that play a critical role in enforcing the SFP(s) that are explicitly identified in the assignment ADV_INT.1.4D. While the entire TSF is to be designed using good engineering principles and result in a modular TSF, the non-SFP-enforcing are not required to adhere as strictly to the coupling and cohesion metrics that are defined for SFP-enforcing modules. The evaluator determines that the design of the TSF and resulting modules are not overly complex and the developer's application of coding standards result in a TSF that is understandable. The intent of this component is for the evaluator to focus on the SFP-enforcing modules and to ensure they exhibit the required characteristics. The evaluator performs limited analysis on the non-SFP-enforcing modules to ensure that these modules are non-SFP-enforcing and that their interaction with SFP-enforcing modules will not adversely affect the SFP-enforcing module's ability to enforce the explicitly assigned SFPs.

812 The primary goal of this component is to ensure the TSF's implementation representation is understandable to facilitate maintenance and analysis (of both the developer and evaluator).

813 Before performing this work unit, the evaluator should have already determined the TSFI that are responsible for enforcing the SFPs in the ST.

Terms, definitions and background

814 The following terms are used in the requirements for software internal structuring. Some of these are derived from the Institute of Electrical and Electronics Engineers *Glossary of software engineering terminology, IEEE Std 610.12-1990.*

815 *modular decomposition:* the process of breaking a system into components to facilitate design and development.

816 *cohesion (*also called *module strength):* the manner and degree to which the tasks performed by a single software module are related to one another; types of cohesion include coincidental, communicational, functional, logical, sequential, and temporal. These types of cohesion are characterised below, listed in the order of decreasing desirability.

817 *functional cohesion:* a module with this characteristic performs activities related to a single purpose. A functionally cohesive module transforms a single type of input into a single type of output, such as a *stack manager* or a *queue manager*.

818 *sequential cohesion:* a module with this characteristic contains functions each of whose output is input for the following function in the module. An example of a sequentially cohesive module is one that contains the functions to write audit records and to maintain a running count of the accumulated number of audit violations of a specified type.

819 *communicational cohesion:* a module with this characteristic contains functions that produce output for, or use output from, other functions within the module. An example of a communicationally cohesive module is an *access check* module that includes mandatory, discretionary, and capability checks.

820 *temporal cohesion:* a module with this characteristic contains functions that need to be executed at about the same time. Examples of temporally cohesive modules include *initialization, recovery,* and *shutdown* modules.

821 *logical (or procedural) cohesion:* a module with this characteristic performs similar activities on different data structures. A module exhibits logical cohesion if its functions perform related, but different, operations on different inputs.

822 *coincidental cohesion:* a module with this characteristic performs unrelated, or loosely related activities.

823 *coupling:* the manner and degree of interdependence between software modules; types of coupling include call, common and content coupling. These types of coupling are characterised below, listed in the order of decreasing desirability

824 *call:* two modules are call coupled if they communicate strictly through the use of their documented function calls; examples of call coupling are data, stamp, and control, which are defined below.

- *data:* two modules are data coupled if they communicate strictly through the use of call parameters that represent single data items.
- *stamp:* two modules are stamp coupled if they communicate through the use of call parameters that comprise multiple fields or that have meaningful internal structures.
- *control:* two modules are control coupled if one passes information that is intended to influence the internal logic of the other.

825 *common:* two modules are common coupled if they share a common data area or a common system resource. Global variables indicate that modules using those global variables are common coupled[1].

826 Common coupling through global variables is generally allowed, but only to a limited degree. For example, variables that are placed into a global area, but are used by only a single module, are inappropriately placed, and should be removed. Other factors that need to be considered in assessing the suitability of global variables are:

The number of modules that modify a global variable: In general, only a single module should be allocated the responsibility for controlling the contents of a global variable, but there may be situations in which a second module may share that responsibility; in such a case, sufficient justification must be provided. It is unacceptable for this responsibility to be shared by more than two modules. (In making this assessment, care should be given to determining the module actually responsible for the contents of the variable; for example, if a single routine is used to modify the variable, but that routine simply performs the modification requested by its caller, it is the calling module that is responsible, and there may be more than one such module). Further, as part of the complexity determination, if two modules are responsible for the contents of a global variable, there should be clear indications of how the modifications are coordinated between them.

The number of modules that reference a global variable: Although there is generally no limit on the number of modules that reference a global variable, cases in which many modules make such a reference should be examined for validity and necessity.

827 *content:* two modules are content coupled if one can make direct reference to the internals of the other (e.g. modifying code of, or referencing labels internal to, the other module). The result is that some or all of the content of one module are effectively included in the other. Content coupling can be thought of as using unadvertized module interfaces; this is in contrast to call coupling, which uses only advertized module interfaces.

828 *call tree:* a diagram that identifies the modules in a system and shows which modules call one another. All the modules named in a call tree that originates with

1. It can be argued that modules sharing definitions, such as data structure definitions, are common coupled. However, for the purposes of this analysis, shared definitions are considered acceptable, but are subject to the cohesion analysis.

(i.e., is rooted by) a specific module are the modules that directly or indirectly implement the functions of the originating module.

829 *software engineering:* the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software. As with engineering practices in general, some amount of judgement must be used in applying engineering principles. Many factors affect choices, not just the application of measures of modular decomposition, layering, and minimisation. For example, a developer may design a system with future applications in mind that will not be implemented initially. The developer may choose to include some logic to handle these future applications without fully implementing them; further, the developer may include some calls to as-yet unimplemented modules, leaving *call stubs*. The developer's justification for such deviations from well-structured programs will have to be assessed using judgement, as well as the application of good software engineering discipline.

830 *complexity:* this is a measure of how difficult software is to understand, and thus to analyse, test, and maintain. Reducing complexity is the ultimate goal for using modular decomposition, layering and minimization. Controlling coupling and cohesion contributes significantly to this goal.

831 A good deal of effort in the software engineering field has been expended in attempting to develop metrics to measure the complexity of source code. Most of these metrics use easily computed properties of the source code, such as the number of operators and operands, the complexity of the control flow graph (*cyclomatic complexity*), the number of lines of source code, the ratio of comments to executable code, and similar measures. Coding standards have been found to be a useful tool in generating code that is more readily understood.

832 This family calls for the evaluator to perform a *complexity analysis* in all components. The developer may be required to support the evaluation team's activities in this area by providing a form of the implementation representation that allows complexity analysis tools to be used to measure some of the properties of the source code.

833 *layering:* the design software such that separate groups of modules (the *layers*) are hierarchically organized to have separate responsibilities such that one layer depends only on layers below it in the hierarchy for services, and provides its services only to the layers above it. Strict layering adds the constraint that each layer receives services only from the layer immediately beneath it, and provides services only to the layer immediately above it.

8.4.1.3 Input

834 The implicit evaluation evidence for this sub-activity is:

a) the ST;

b) the low-level design;

c) the implementation representation;

d) the architectural description.

8.4.1.4 Evaluator actions

835 This sub-activity comprises three CC Part 3 evaluator action elements:

a) ADV_INT.1.1E;

b) ADV_INT.1.2E;

c) ADV_INT.1.3E.

8.4.1.4.1 Action ADV_INT.1.1E

ADV_INT.1.1C

CC Text: ***The software architectural description shall identify the SFP-enforcing and non-SFP-enforcing modules.***

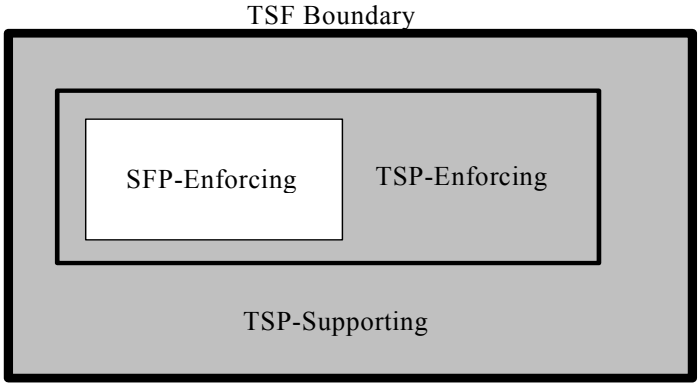
ADV_INT.1-1 The evaluator *shall check* the software architectural description to determine that it identifies the SFP-enforcing and non-SFP-enforcing modules.

836 The software architectural description clearly identifies which modules are SFP-enforcing and which modules are non-SFP-enforcing modules. The evaluator ensures that all modules in the software architectural description are designated as either an SFP-enforcing or non-SFP-enforcing modules.

837 The non-SFP-enforcing portions of the TSF consist of the TSP-supporting modules and TSP-enforcing modules that do not play a role in the enforcement of the SFP(s) identified in ADV_INT.1.4D as depicted in the figure below, where is this example, non-SFP-enforcing is everything in the TSF other than the SFP-enforcing functions.

838 The developer is required to identify the modules that are SFP-enforcing and implicitly the remaining modules, which will be non-SFP-enforcing. The SFP-enforcing modules are those modules that interact with the module or modules that provide the TSFI for that SFP with justified exceptions.

839



ADV_INT.1-2 The evaluator *shall examine* the coupling analysis to determine that the software architectural description accurately identifies the modules as SFP-enforcing or non-SFP-enforcing.

840 The parts of the TSF that implement an SFP (i.e., access control, information flow control, or any other policies identified in ADV_INT.1.4D of the PP or ST) are those modules that interact with the module or modules that provide the TSFIs for that SFP, with justified exceptions.

841 The evaluator uses the coupling analysis and global analysis to determine the interactions of modules. The coupling analysis may include a call tree, identify functions that are passed to other functions via pointers, or in a message passing system identifies the messages that are passed between modules. The global variable analysis identifies the global variables used in the TSF, and identifies the modules and their mode of reference (e.g., write, read) to each global variable.

842 The evaluator uses the coupling analysis provided with the software architectural description and the module descriptions provided as part of ADV_LLD to ensure that all of the modules that interact with the module that exports the TSFI are correctly identified as SFP-enforcing and the remaining modules are identified as non-SFP-enforcing.

843 The evaluator uses the global variable analysis to determine if modules that are identified as non-SFP-enforcing modules set or write to SFP-enforcing global variables that are read by an SFP-enforcing module. Modules that write to SFP-enforcing global variables (variables that are related to an SFR for the SFPs identified in ADV_INT.1.4D) and are read by the module or modules that provide the TSFIs for that SFP, or read by other modules that are deemed SFP-enforcing because of their interactions with SFP-enforcing modules, are deemed SFP-enforcing. The use of global variables potentially plays as critical a role as making an explicit call to a module. The intent is that all of the modules that play a SFR related role (as opposed to modules that provide infrastructure support, such as scheduling, reading binary data from the disk) in enforcing an SFP are identified as SFP-enforcing.

844 For example, module A presents the open() TSFI, which plays a critical role in enforcing the FDP_ACC policy, and calls functions in modules B, C and D. Module B is responsible for making the access control decision (thereby being SFP-enforcing), which in turn calls modules X and Y. Module X is responsible for sorting the access control list, and module Y is called to allocate memory for module B. If the access control policy has a rule concerning the ordering of access control lists, then module X is considered SFP-enforcing. If there is no rule regarding ordering of access control lists (ordering of access control lists makes no difference), then module X can be considered non-SFP-enforcing. If the only role module Y plays is the allocation of memory for module B, then module Y could be considered non-SFP-enforcing (with respect to the FDP_ACC requirements, if a residual data protection policy was identified as an SFP in this component, then module Y may be SFP-enforcing).

845 The evaluator could perform this work unit in conjunction with work unit ADV_INT.1-4.

846 The evaluator should perform the work unit ADV_INT.1-6 to ensure the coupling analysis is accurate and complete and the work unit ADV_INT.1-7 to ensure the global variable analysis is complete and accurate before performing this work unit.

ADV_INT.1.2C

CC Text: ***The TSF modules shall be identical to those described by the low-level design (ADV_LLD.1.4C).***

ADV_INT.1-3 The evaluator *shall check* the software architectural description to determine that the identification of modules is identical to the modules that are described in the low-level design.

847 The evaluator determines that the modules described in the low-level design are identical to those presented in the software architectural description. This is confirmed by the evaluators in comparing the two sets of evidence.

ADV_INT.1.3C

CC Text: ***The software architectural description shall provide a justification for the designation of non-SFP-enforcing modules that interact with the SFP-enforcing modules.***

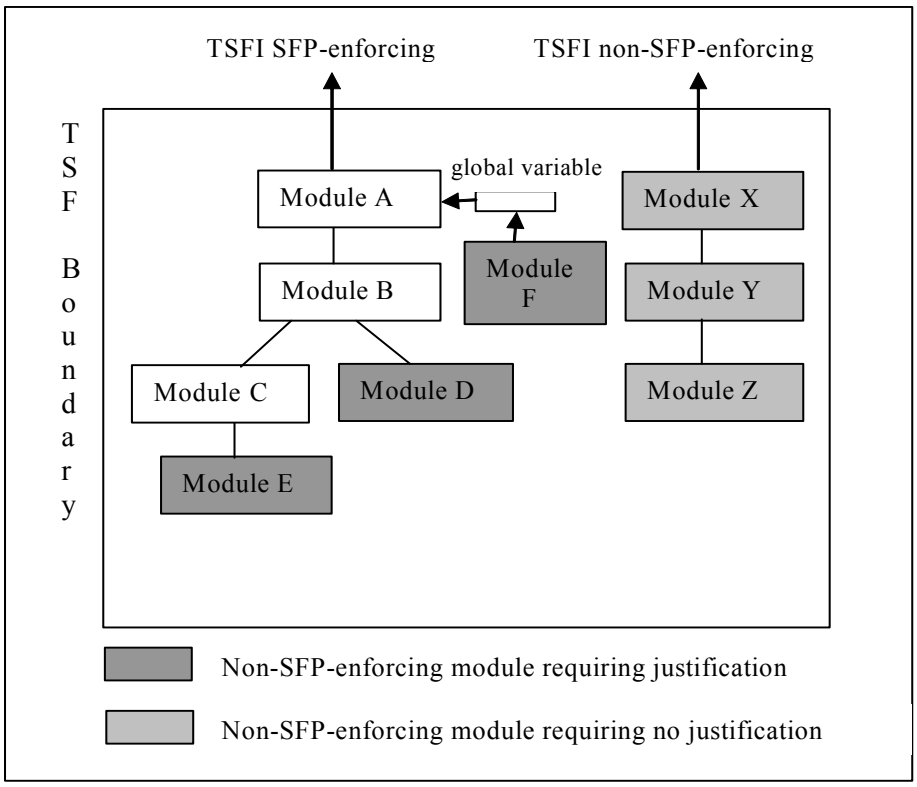
ADV_INT.1-4 The evaluator *shall examine* the software architectural description to determine that the developer provided justification for the designation of non-SFP-enforcing modules that interact with the SFP-enforcing modules accurately reflects the module's role in policy enforcement.

848 The evaluator ensures that the justification for why a module is non-SFP-enforcing even though it interacts with an SFP-enforcing module, and therefore is relied upon by an SFP-enforcing module to perform some function, is reasonable. The justification should discuss that the non-SFP-enforcing module plays no direct role in satisfying an SFR or portions of an SFR. A module that interacts an SFP-

enforcing module is justifiably designated a non-SFP-enforcing module if it is only providing infrastructure support.

849 Modules that provide TSFIs that are not SFP-enforcing do not require justification for their designation, nor do the modules in the call tree for such modules. For example, Modules X, Y, and Z in the example below require no justification. From previous work units (e.g., ADV_FSP) it has already been determined that the TSFI is non-SFP-enforcing and therefore the modules that implement or provide services to implement the TSFI are also non-SFP-enforcing. On the other hand, Modules D and E exist in the call tree for a module that exports a TSFI that is SFP-enforcing and therefore those modules potentially are SFP-enforcing and therefore require a justification (e.g., provides infrastructure support that allocates memory to a module) as to why they are non-SFP-enforcing. Module F writes to a global variable that is read by SFP-enforcing modules, therefore it also is potentially an SFP-enforcing module.

850



ADV_INT.1.4C

CC Text: ***The software architectural description shall describe the process used for modular decomposition.***

ADV_INT.1-5 The evaluator *shall examine* the software architectural description to determine that the developer's process for modular decomposition results in a modular TSF design.

851 Modules interact by providing services to one another, or by cooperating in achieving an overall goal.

852 The evaluator assesses the developer's design methodology to determine that the TSF design incorporates sound engineering principles to decompose the TSF into modules. Modules should be designed to include only related functions, to have a single, well-defined purpose, and to avoid unnecessary interactions with other modules. The methodology should describe what are acceptable forms of communication between modules and under what circumstances discouraged forms of coupling and cohesion are considered acceptable. The methodology should also address complexity and what steps are taken to ensure modules are not overly complex. The intent here is that the developer's process should cover/ address the developer action elements for the ADV_INT.1 requirements and this process provides guidance to the developers on how to code their modules.

853 The methodology provided by the developer provides the evaluator with an understanding of the developer's approach to designing (or restructuring of) the TSF and aids the evaluator in performing other work units for this component.

ADV_INT.1.5C

CC Text: ***The software architectural description shall describe how the TSF design is a reflection of the modular decomposition process.***

ADV_INT.1-6 The evaluator *shall examine* the software architectural description to determine that the TSF's resulting design is an accurate reflection of the developer's decomposition process.

854 The evaluator compares the developer's documented process for decomposition presented in the architectural description with the actual implementation to determine if the developer followed their process. The evaluator will need to perform this work unit in conjunction with other work units. In fact, the evaluator should understand the process the developer has employed for decomposing their TSF into modules and while performing other work units ensure that process was followed.

ADV_INT.1.6C

CC Text: ***The software architectural description shall include the coding standards used in the development of the TSF.***

ADV_INT.1-7 The evaluator *shall examine* the coding standards to determine that contain the necessary information for the programming language(s) used to ensure the implementation of the TSF is consistent and readily understandable.

855 Coding standards play an important role in developing an implementation representation that is easily understood. Coding standards address many aspects of programming including format and programming logic standards. The developer is not required to adhere to industry standard coding standards (e.g., ANSI, IEEE) so as to allow for flexibility (and standards may not exist for all employed programming languages).

856 The evaluator ensures the developer's coding standards are complete, consistent, and address the important aspects of the employed programming languages. The evaluator determines completeness by assessing if the coding standards are followed the result will be an easily readable and understandable implementation representation.

ADV_INT.1-8 The evaluator *shall examine* the implementation representation to determine that the developer adhered to the coding standards.

857 The evaluator ensures that the development of implementation representation adheres to the coding standards. A developer may have more than one coding standard for different components of the TOE, and this is acceptable, as long as each standard contains the necessary information. This is accomplished by determining that the format prescribed by the coding standards (e.g., comments, naming conventions, indentation) is exhibited in the implementation representation. The evaluator ensures that the guidelines for programming logic (e.g., types of loops, control flow, error handling, the use of pointers) are exhibited in the implementation representation. The evaluator determines that other aspects of code development that are defined in the coding standards are followed by the developer.

ADV_INT.1.7C

CC Text: ***The software architectural description shall provide a justification, on a per module basis, of any deviations from the coding standards.***

ADV_INT.1-9 The evaluator *shall examine* the architectural description to determine if the justification for deviations from the coding standards for each module is reasonable.

858 There may be acceptable occurrences where the implementation representation does not follow the coding standards. A justification that is solely based on the use of third-party code or code that was inherited from previous life-cycles of the TOE is not acceptable. Typically, deviations from the prescribed formatting aspects of the coding standards are not justifiable. Acceptable justification for deviation from the coding standards would generally be related to performance or efficiency reasons, and for compatibility (backward or future).

ADV_INT.1-10 The evaluator *shall examine* the implementation representation to determine if the deviations from the coding standards for each module do not make the offending modules overly complex.

859 The evaluator assesses the modules that have been identified in the architectural description as deviating from the coding standards to ensure the deviations do not make the module too complex to understand the module's processing and interactions in a reasonable amount of time (i.e., no more than eight hours). The developer's complexity analysis addresses the programmatic complexity of the module, but here, the evaluator assesses the modules not adhering to the coding standards to ensure the code is readable and readily understood.

ADV_INT.1.8C

CC Text: ***The software architectural description shall include a coupling analysis that describes intermodule coupling for the SFP-enforcing modules.***

ADV_INT.1-11 The evaluator *shall examine* the coupling analysis to determine that it is a complete and accurate reflection of the implementation representation.

860 The evaluator ensures that the coupling analysis correctly identifies and describes the method of all of the interactions with other modules by examining the implementation representation. The coupling analysis includes any explicit calls made by functions in a module to other modules, the passing of pointers to function to another function, the use of global variables, shared memory, message passing, or any other means that functions may use to communicate with other modules.

861 The evaluator also uses the implementation representation to ensure modules do not reference global variables that are not contained in the coupling analysis and that the references modules make (e.g., read, write) are consistent with the references specified in the coupling analysis.

862 The evaluator makes an independent assessment of the type of coupling that exists between modules and ensures that they arrive at the same conclusion as the presented in the developer's analysis.

ADV_INT.1-12 The evaluator *shall examine* the coupling analysis to ensure the SFP-enforcing modules exhibit only call or common coupling, with limited exceptions.

863 Once the evaluator has completed work unit **ADV_INT.1-11** and has determined that the coupling analysis is complete and accurate, the evaluator determines that the instances of coupling other than call or common coupling are justified by the software architectural description. The evaluator determines if the justifications are reasonable, and therefore allowable in a separate work unit.

ADV_INT.1.9C

CC Text: ***The software architectural description shall provide a justification, on a per module basis, for any coupling or cohesion exhibited by SFP-enforcing modules, other than those permitted.***

ADV_INT.1-13 The evaluator *shall examine* the software architectural description to determine if the justification for coupling and cohesion other than the ones permitted in the SFP-enforcing modules are reasonable.

864 The evaluator judges the developer's rationale for modules that exhibit coupling and cohesion other than those allowed and determines if the justifications are reasonable. The justification is provided for each module that is not conformant to the allowed forms of coupling and cohesion.

865 Reasons for justification may include efficiency or performance and the evaluator uses their judgment to determine that the justification is acceptable and that the disallowed forms of cohesion or coupling are not pervasive. A justification that is solely based on the use of third-party code or code that was inherited from previous life-cycles of the TOE is not acceptable.

ADV_INT.1.10C

CC Text: ***The software architectural description shall provide a justification, on a per module basis, that the SFP-enforcing modules are not overly complex.***

ADV_INT.1-14 The evaluator *shall examine* the software architectural description to determine that the justification(s) that the SFP-enforcing module(s) are not overly complex is reasonable.

866 The evaluator assesses the justifications to ensure the developer's design of the SFP-enforcing modules and resulting implementation representation is not overly complex. The developer's justification could comprise of results of a tool that is run that measures the complexity and accompanying prose that discusses the tool's result in the context of the developer's design methodology and resulting implementation. The justification could be a prose description that provides evidence that the TSF is not overly complex. If the developer's implementation adheres to the allowed forms of coupling and cohesion, and if developers followed acceptable coding standards, the developer

867 No matter what method is chosen the metrics that are employed in the analysis must be identified and described.

8.4.1.4.2 Action ADV_INT.1.2E

CC Text: ***The evaluator shall perform a cohesion analysis for the modules that substantiates the type of cohesion claimed for a subset of SFP-enforcing modules.***

ADV_INT.1-15 The evaluator *shall examine* the implementation representation to determine that the type of cohesion exhibited by SFP-enforcing modules is consistent with that claimed in the software architectural description for the SFP-enforcing modules.

868 Cohesion is a measure of the strength of relationship between the activities performed by a module. Cohesion ranges from the most desirable form, functional cohesion, to the least desirable form, coincidental cohesion. Allowable forms of cohesion include functional, sequential, and communicational cohesion;

coincidental cohesion is unacceptable. Temporal cohesion is acceptable only for special-purpose use, such as initialization, shutdown, and error recovery. Logical cohesion may be acceptable, based on the argument for its justification.

869 The evaluator uses the implementation representation and low-level design to determine the purpose of the modules and the relationship the functions have within a module to achieve that purpose. The processing performed by the functions in a module and the service they provide are an indication of the design in the grouping of functions and what type of cohesion is exhibited.

870 The evaluator performs an analysis to determine the type of cohesion exhibited by the modules and ensures that the evaluator's findings are consistent with the developer's claim in the software architectural description. It is important to note that there may be some subjectivity involved in making a determination of the type of cohesion exhibited (e.g., sequential vs. temporal), but the type of cohesion that is deemed unacceptable for SFP-enforcing modules (e.g., coincidental, logical) is more clear cut.

8.4.1.4.3 Action ADV_INT.1.3E

CC Text: ***The evaluator shall perform a complexity analysis for a subset of TSF modules.***

ADV_INT.1-16 The evaluator *shall examine* the implementation representation to determine that a subset of the TSF modules are not overly complex.

871 The non-SFP-enforcing modules in the TOE are not held to as strict of a standard as the SFP-enforcing modules. A developer may have modules that exhibit coincidental or local cohesion without a justification. The non-SFP-modules may also exhibit the undesirable forms of coupling (e.g., content) without justification. This allows developers to use third-party software or portions of previous versions of the TOE without having to restructure the implementation representation. However, these modules must be non-SFP-enforcing and not overly complex.

872 A metric that simply relies on the numbers of lines of code, or the percentage of comments and statements is considered a weak complexity measure and does provide a good metric of software complexity. More meaningful metrics consider the number of unique operators and operands in the program, or which measures the amount of unstructured decision logic in software by examining cyclomatic complexity.

873 The evaluator determines the complexity of the implementation representation by measuring the length of time it takes the evaluator to understand the logic in a module and the module's relationship with other modules. When using the low-level design and architectural description, the evaluator should understand a module's implementation and interactions in a reasonable amount of time. The measurement of time is subjective and is based on the evaluator's experience and skill in reading and understanding source code. It is assumed that the evaluator has knowledge of computer programming, and is familiar with the programming language used by the developer. Given this minimum level of expertise, the

amount of time it takes to understand a module's implementation should not exceed eight hours.

874 The evaluator bases their determination of complexity as in other work units pertaining to complexity. The intent is not for the evaluator to spend a lot of time analyzing the non-SFP-enforcing modules. The evaluator should be able to easily determine that a non-SFP-enforcing module is performing infrastructure support, or enforces a policy that has not be explicitly assigned in this component, and that these modules cannot adversely impact the SFP-enforcing modules ability to enforce their respective policies.

8.5 Evaluation of low-level design

8.5.1 Sub-activity ADV_LLD.1

8.5.1.1 Objectives

875 The objective of this sub-activity is to use the low-level design as a guide to reading the source code, and to use the low-level design and source code to reach an understanding of the security enforcing aspects of the TSF identified by the developer in detail sufficient such that the evaluator has some confidence that the functional specification, high-level design, and architectural design and description are complete and accurate, and that the testing adequately covers the security requirements placed on the TSF.

8.5.1.2 Application Notes

876 There are two types of activity that the evaluators must undertake with respect to the low-level design. First, the evaluators determine that the developer has provided documentation that conforms to the requirements of ADV_LLD.1. Beyond that, the evaluators must analyze the low-level design information provided for the security-enforcing modules to understand how the system is implemented, and with that knowledge ensure that the TSFI in the functional specification are adequately described, and that the test information adequately tests the TSF.

877 It is important to note that while the developer is obligated to provide a complete description of the TSF (although security-enforcing modules will have more detail than the security-supporting modules), **the evaluator is expected to use their judgment in performing their analysis.** While the evaluator is expected to look at every module, the detail to which they examine each module may vary. The evaluator analyzes each module in order to gain enough understanding to determine the effect of the functionality of the module on the security of the system, and the depth to which they need to analyze the module may vary depending on the module's role in the system. An important aspect of this analysis is that the evaluator should use the other documentation provided (TSS, functionality specification, high-level design, architectural design, architectural description, and the implementation representation) in order to determine that the functionality that is described is correct, and that the designation of security-supporting modules is supported by their role in the system architecture.

8.5.1.3 Input

878 The evaluation evidence for this sub-activity is:

a) the low-level design;

b) the ST;

c) the functional specification;

d) the high-level design;

e) the architectural design;

f) the architectural description;

g) the implementation representation; and

h) the correspondence demonstration.

8.5.1.4 Evaluator actions

879 This sub-activity comprises two CC Part 3 evaluator action elements:

a) ADV_LLD.1.1E;

b) ADV_LLD.1.2E.

8.5.1.4.1 Action ADV_LLD.1.1E

ADV_LLD.1.1C

CC Text : ***The presentation of the low-level design shall be informal.***

ADV_LLD.1-1 The evaluator *shall examine* the low-level design to determine that it contains all necessary informal explanatory text.

880 If the entire low-level design is informal, this work unit is not applicable and is therefore considered to be satisfied.

881 Supporting narrative descriptions are necessary for those portions of the low-level design that are difficult to understand only from the semiformal or formal description (for example, to make clear the meaning of any formal notation).

ADV_LLD.1.2C

CC Text : ***The presentation of the low-level design shall be separate from the implementation representation.***

ADV_LLD.1-2 The evaluator *shall examine* the low-level design to determine that it is presented separate from the implementation representation.

882 The evaluator analyses the low-level design to ensure that it is not the implementation representation itself, nor that it is "interspersed" with the implementation representation. For example, a "module" that is a source code file with comments is not acceptable. The low-level design in part is intended as a guide to understanding the implementation representation; it is not the implementation representation itself.

ADV_LLD.1.3C

CC Text : ***The low-level design shall be internally consistent.***

ADV_LLD.1-3 The evaluator *shall examine* the presentation of the low-level design to determine that it is internally consistent.

883 In analysing the low-level design, the evaluator gains an understanding of how the TOE "works." The evaluator should examine module dependencies, in both the functional call sense and the global data sense. If Module A claims to use Module B for some function, yet the algorithmic description of Module B does not describe that function, then an inconsistency may be present. Likewise, if there is a global data area used for a particular purpose, and the algorithmic description of Module A seems to indicate that data from that global data area is necessary but it is not specified as being used by Module A, then an inconsistency should be noted.

884 It should be noted that because security-supporting modules contain a lesser degree of detail than do security-enfocing modules, the consistency analysis will be more complete for security-enfocing modules than for the security-supporting modules. The level of analysis perform should be documented in the ETR sections produced for this work unit.

885 Part of consistency is that the modules are correctly identified as security-supporting and security-enforcing. In performing the other work units of this class for the security-supporting modules, the evaluators determine if the information presented for the security-supporting modules supports the claim that they are security supporting. For example, in performing work units ADV_LLD.1-13 and ADV_LLD.1-14 the evaluator should determine that the descriptions for the security-supporting modules do not indicate any functionality that might be security-enforcing.

886 For general guidance on consistency analysis see Chapter 14.3.

ADV_LLD.1.4C

CC Text : ***The low-level design shall identify and describe data that are common to more than one module, where any of the modules is a security-enforcing module.***

ADV_LLD.1-4 The evaluator *shall examine* the low-level design to determine that it identifies and describes the global data that are common to a security-enforcing module and at least one other module.

887 If the implementation does not make use of global data, then this work unit is not applicable and considered to be satisfied.

888 Global data are those data that are used by more than one module, yet are not passed as formal parameters or messages to a module. Data that are global only among security-supporting modules are not analyzed as part of this work unit. Note that the criterion is that the data are *used*, as opposed to just "being available." The intent is that the evaluator perform this work unit for global data that are used (read or written) by security-enfocing modules, even if all of the other users of those data are security-supporting modules.

889 The evaluator determines that global data are identified and described much like parameters of an interface. The evaluator should be able to tell from the description of the global data what their role in the system design and implementation is. The evaluator should also refer to the global variable analysis performed as part of the coupling analysis in the architectural description document (ADV_INT.1) to determine whether the identification and description of the global data is complete.

890 The evaluator examines other information available (functional specification, high-level design, implementation representation) to determine that the global data described seems complete.

ADV_LLD.1.5C

CC Text : ***The low-level design shall describe the TSF in terms of modules, designating each module as either security-enforcing or security-supporting.***

ADV_LLD.1-5 The evaluator *shall examine* the low-level design to determine that it describes the TSF in terms of modules, and each is designated as either security-enforcing or security-supporting.

891 The term *module* is used in this family by the CC to denote a less abstract entity than a subsystem. A module is a relatively small (compared to the size of the TOE) entity that exhibits coupling and cohesion properties. See ADV_INT for further discussion on coupling and cohesion.

892 "Security enforcing" is used to to describe modules that implement at least one security-enforcing TSFI. Since a TSF module must be either security enforcing or security supporting, the security-supporting subsystems are merely those that are not security enforcing.

In determining whether the designation of a module as security enforcing or security supporting is accurate, the evaluator should examine the TSFI and subsystems that the module supports (if not evident from the low-level design itself, this information should be contained in the ADV_RCR analysis provided by the developer). If any of the TSFI or subsystems supported by a module are security enforcing, then that module should be designated as security enforcing in the low-level design.

ADV_LLD.1.6C

CC Text : ***The low level design shall describe each security-enforcing module in terms of its purpose, interfaces, return values from those interfaces, called interfaces to other modules, and global variables.***

ADV_LLD.1-6 The evaluator *shall examine* the low-level design to determine that the description of the purpose of each security-enforcing module is complete and accurate.

893 The purpose a module provides is a short description indicating what function the module is fulfilling. The evaluator should be able to obtain a general idea of what the module's function is in the architecture from the description. In order to assure

the description is complete, the evaluator uses other information provided about the module in the low-level design (interfaces, algorithmic description, etc.) to ensure that the major functionality is reflected in the description. The evaluator determines accuracy by ensuring that the description of the functionality matches the information contained in the module description.

894 Because the modules are at such a low level, it may be difficult determine completeness and accuracy impacts from other documentation, such as administrative guidance, the high-level design, the functional specification, the architectural design, or the architectural description. However, the evaluator uses the information present in those documents to the extent possible to help ensure that the function is accurately and completely described. This analysis can be aided by the evidence produced in ADV_RCR, which maps the TSFI in the functional specification to the subsystems of the TSF, and maps the subsystems of the TSF to the modules identified in the low-level design.

895 The evaluator should also examine the source code described by the low-level design for the module. While it is not expected that the evaluator necessarily examine the source code for each security-enforcing module, the evaluator should sample the source code related to modules that describe different portions of the security architecture (auditing, I&A, etc.) to help make the case that the LLD is a complete and accurate reflection of the implementation.

ADV_LLD.1-7 The evaluator *shall examine* the low-level design to determine that the description of the interfaces presented by each module contain an accurate and complete description of the parameters passed through the interface, the invocation sequence for each interface, and any values returned directly by the interface.

896 The interfaces of a module are those interfaces used by other modules to invoke the functionality provided. Interfaces are described in terms of how they are invoked, and any values that are returned. This description would include a list of parameters, and descriptions of these parameters. If a parameter were expected to take on a set of values (e.g., a “flag” parameter), the complete set of values the parameter could take on that would have an effect on module processing would be specified. Likewise, parameters representing data structures are described such that each field of the data structure is identified and described. Note that different programming languages may have additional “interfaces” that would be non-obvious; an example would be operator/function overloading in C++. This “implicit interface” in the class description would also be described as part of the low-level design. Note that although a module could present only one interface, it is more common that a module presents a small set of related interfaces.

897 The invocation sequence is a programming-reference-type description that one could use to correctly invoke a module’s interface if one were writing a program to make use of the module’s functionality through that interface.

898 Values returned directly by the interface refer to values that either passed through parameters or messages, or values that the function call itself returns in the style of a “C” program function call.

899 In order to assure the description is complete, the evaluator uses other information provided about the module in the low-level design (e.g., algorithmic description, global data used) to ensure that it appears all data necessary for performing the functions of the module is presented to the module, and that any values that other modules expect the module under examination to provide are identified as being returned by the module. The evaluator determines accuracy by ensuring that the description of the processing matches the information listed as being passed to or from an interface.

900 Because the modules are at such a low level, it may be difficult determine completeness and accuracy impacts from other documentation, such as administrative guidance, the high-level design, the functional specification, the architectural design, or the architectural description. However, the evaluator uses the information present in those documents to the extent possible to help ensure that the interfaces are accurately and completely described. This analysis can be aided by the evidence produced in ADV_RCR, which maps the TSFI in the functional specification to the subsystems of the TSF, and maps the subsystems of the TSF to the modules identified in the low-level design.

901 The evaluator should also examine the source code described by the low-level design. While it is not expected that the evaluator necessarily examine the source code for *each* security-enforcing module, the evaluator should sample the source code related to modules that describe different portions of the security architecture (auditing, I&A, etc.) to help make the case that the LLD is a complete and accurate reflection of the implementation.

ADV_LLD.1-8 The evaluator *shall examine* the low-level design to determine that the description of the interfaces used by each security-enforcing module are uniquely and completely identified.

902 Interfaces used by each security-enforcing module must be identified in a unique manner such that it can be determined by the evaluator which module is being invoked by the module being described. Note that this applies to all interfaces used by a security-enforcing module, not just those interfaces presented by other security-enforcing modules.

903 In order to assess completeness, the evaluator uses other information provided about the module in the low-level design (e.g., algorithmic description) to ensure that it appears all functionality that is not supplied directly by the code associated with that module is provided by an identified (external) module. For instance, if the algorithmic description of a module is discussing the manipulation of certain data, and from one line to the next it appears that the data are “suddenly” sorted, the evaluator might question whether a “sort module” interface was invoked to accomplish this.

904 The evaluator should also examine the implementation representation described by the low-level design for the security-enforcing modules. While it is not expected that the evaluator necessarily examine the source code for *each* security-enforcing module, the evaluator should sample the source code related to modules that describe different portions of the security architecture (auditing, I&A, etc.) to help

make the case that the LLD is a complete and accurate reflection of the implementation.

ADV_LLD.1-9 The evaluator *shall examine* the low-level design to determine that the description of the interfaces used by each security-enforcing module are of sufficient detail to determine the algorithmic purpose of invoking the interface, and the expected results of invoking the interface.

905 It must also be clear from the low-level design the algorithmic reason the invoking module is being called. For instance, if Module A is being described, and it uses Module B's bubble sort routine, an inadequate algorithmic description would be "Module A invokes the double_bubble() interface in Module B to perform a bubble sort." An adequate algorithmic description would be "Module A invokes the double_bubble routine with the list of access control entries; double_bubble() will return the entries sorted first on the username, then on the access_allowed field according the following rules..." The low-level design must provide enough detail so that it is clear what effects Module A is expecting from the bubble sort interface. Note that one method of presenting these called interfaces is via a call tree, and then the algorithmic description can be included in the algorithmic description of the module. If the call-tree approach is used, the analysis of the work associated with the previous work unit might be made easier.

906 The evaluator should also examine the implementation representation described by the low-level design for the security-enforcing modules. While it is not expected that the evaluator necessarily examine the source code for *each* security-enforcing module, the evaluator should sample the source code related to modules that describe different portions of the security architecture (auditing, I&A, etc.) to help make the case that the LLD is complete and accurate with respect to the algorithmic reason for calling an interface.

ADV_LLD.1-10 The evaluator *shall examine* the low-level design to determine that the description of the global data used by each security-enfocing module is complete and accurate.

907 If the implementation does not make use of global data, then this work unit is not applicable and considered to be satisfied.

908 A module "uses" global data if it either reads or write the data. In order to assure the description of global data used is complete, the evaluator uses other information provided about the module in the low-level design (interfaces, algorithmic description, etc.), as well as the description of the particular set of global data. For instance, the evaluator could first determine the processing the module performs by examining its function and interfaces presented (particularly the parameters of the interfaces). They could then check to see if the processing appears to "touch" any of the global data areas identified in the low-level design. The evaluator then determines that, for each global data area that appears to be "touched", that global data area is listed as "used" by the module the evaluator is examining.

909 The evaluator determines accuracy by ensuring that, for each global data area listed as being used by the module, the other information associated with the module (algorithmic description, interfaces, etc.) is consistent with the claimed usage. For

example, if a certain module claimed to use a global process table structure, but the other information about the module claimed that it performed path-name traversal look-up functions, the evaluator should question the accuracy of the claim that the process table structure is used.

910 The evaluator also examines the portion of the architectural description (ADV_INT) pertaining to global variables (i.e., the coupling analysis) to determine that the description in the LLD is accurate in complete; there should be no discrepancies between the information in the architectural description that is not also in the module description in the LLD.

911 The evaluator should also examine the implementation representation described by the low-level design for the module. While it is not expected that the evaluator necessarily examine the source code for each security-enforcing module, the evaluator should sample the source code related to modules that describe different portions of the security architecture (auditing, I&A, etc.) to help make the case that the LLD completely and accurately describes the global variables.

912 ADV_LLD.1.7C

CC Text : ***For each security-enforcing module, the low level design shall provide an algorithmic description detailed enough to represent the TSF implementation.***

ADV_LLD.1-11 The evaluator *shall examine* the low-level design to determine that the algorithmic description of each security-enforcing module is complete and accurate.

913 This work unit should be performed in association with ADV_LLD.1-12. This work unit deals with quality of the algorithmic description, while ADV_LLD.1-12 deals with the level of detail associated with the algorithmic description.

914 The algorithmic description is complete if it describes (in an algorithmic fashion) all of the functionality performed by the module. This description should detail how the module inputs are manipulated to produce the desired system effect, using the information provided about the parameters to the interface being invoked, as well as any calls made to other modules. The evaluator should examine the other evidence presented about the module (global data use, interfaces, function) to ensure that the description appears to be complete.

915 Interfaces used by each security-enforcing module are identified in a unique manner such that it can be determined by the evaluator which module is being invoked by the module being described. Note that this applies to all interfaces used by a security-enforcing module, not just those interfaces presented by other security-enforcing modules.

916 It must also be clear from the low-level design the algorithmic reason the invoking module is being called. For instance, if Module A is being described, and it uses Module B's bubble sort routine, an inadequate algorithmic description would be "Module A invokes the double_bubble() interface in Module B to perform a bubble sort." An adequate algorithmic description would be "Module A invokes the double_bubble routine with the list of access control entries; double_bubble() will

return the entries sorted first on the username, then on the access_allowed field according the following rules...” The low-level design must provide enough detail so that it is clear what effects Module A is expecting from the bubble sort interface. Note that one method of presenting these called interfaces is via a call tree, and then the algorithmic description can be included in the algorithmic description of the module.

917 Because the algorithmic description is at such a low level of detail, it may be difficult determine completeness and accuracy impacts from higher-level documentation, such as administrative guidance, the high-level design, the functional specification, the architectural design, or the architectural description. However, the evaluator uses the information present in those documents to help ensure that the algorithmic description is accurately and completely described. This analysis can be aided by the evidence produced in ADV_RCR, which maps the TSFI in the functional specification to the subsystems of the TSF, and maps the subsystems of the TSF to the modules identified in the low-level design.

918 The description is accurate if it correctly reflects the implementation. While the evaluator can do some of this analysis by examining other information about the module (e.g., functional specification, high-level design), the evaluator examines the implementation representation to gain confidence that the algorithmic descriptions for the modules is accurate. While it is not expected that the evaluator necessarily examine the source code for *each* security-enforcing module, the evaluator should sample the source code related to modules that describe different portions of the security architecture (auditing, I&A, etc.) to help make the case that the LLD is a complete and accurate description of the implemented algorithm.

ADV_LLD.1-12 The evaluator *shall examine* the low-level design to determine that the algorithmic detail for each security-enforcing module is representative of the actual TSF implementation.

919 The algorithmic description of a security-enforcing module describes in an algorithmic fashion the implementation of the module. This can be done in pseudo-code, through flow charts, or other semi-formal presentation methods. It discusses how the parameters to the interface, global data, and called modules are used to accomplish the function of the module. It notes changes to global data, system state, and return values produced by the module. A description of a module in the low-level design should be of sufficient detail so that one could create an implementation of the module from the low-level design, and that implementation would be identical to the actual TSF implementation in terms of the interfaces presented and used by the module, and the internals of the module would be algorithmically identical. For instance, the low-level design may describe a block of processing that is looped over a number of times. The actual implementation may be a **for** loop or a **do** loop, both of which could be used to implement the algorithm. Conversely, if a module’s actual implementation performed a bubble sort, it would be inadequate for the algorithmic description to specify that the module “performed a sort”; it would have to describe the sort of sort that was being performed in an algorithmic fashion. See Knuth if you are inclined to argue about “algorithmic” means.

920 As previously indicated, the evaluator should sample the implementation representation for modules to ensure the description in the LLD reflects the implementation in the appropriate amount of detail. It should be fairly easy for the evaluators to understand all portions of a module's implementation representation if the LLD is written in sufficient detail.

921 ADV_LLD.1.8C

CC Text : ***The low level design shall describe each security-supporting module in terms of its purpose and interaction with other modules.***

ADV_LLD.1-13 The evaluator *shall examine* the low-level design to determine that the description of the purpose of each security-supporting module is complete and accurate.

922 The purpose a module provides is a short description indicating what function the module is fulfilling. The evaluator should be able to obtain a general idea of what the module's function is in the architecture from the description. In order to assure the description is complete, the evaluator uses the information provided about the module's interactions with other modules to assess whether the reasons for the module being called are consistent with the module's purpose. If the interaction description contains functionality that is not apparent from, or in conflict with, the module's purpose, the evaluator needs to determine whether the problem is one of accuracy or of completeness.

923 Because the modules are at such a low level, it may be difficult determine completeness and accuracy impacts from other documentation, such as administrative guidance, the high-level design, the functional specification, the architectural design, or the architectural description. However, the evaluator uses the information present in those documents to the extent possible to help ensure that the function is accurately and completely described. This analysis can be aided by the evidence produced in ADV_RCR, which maps the TSFI in the functional specification to the subsystems of the TSF, and maps the subsystems of the TSF to the modules identified in the low-level design.

ADV_LLD.1-14 The evaluator *shall examine* the low-level design to determine that the description of a security-supporting module's interaction with other modules is complete and accurate.

924 A module's interaction with other modules can be captured in many ways. The intent for the low-level design is to allow the evaluator to understand (in part through analysis of module interactions) the role of the security-supporting modules in the architecture. Understanding of this role will allow the evaluator to independantly validate the claim that the module is indeed only security-supporting.

925 A module's interaction with other modules goes beyond just a call-tree-type document. The interaction is described from a functional perspective of why a module interacts with other modules. The module's purpose describes what functions the module provides to other modules; the interactions should describe what the module depends on from other modules in order to accomplish this function.

8.5.1.4.2 Action ADV_LLD.1.2E

CC Text: ***The evaluator shall determine that the low-level design is an accurate and complete instantiation of all TOE security functional requirements with the exception of FPT_SEP and FPT_RVM.***

ADV_LLD.1-15 The evaluator *shall examine* the low-level design to determine that it is a complete instantiation of the user-visible TOE security functional requirements.

926 To ensure that all ST security functional requirements are covered by the low-level design, the evaluator may construct a map between the TOE security functional requirements and the low-level design. This map will likely be from a functional requirement to a set of modules. Note that this map may have to be at a level of detail "below" the component or even element level of the requirements, because of operations (assignments, refinements, selections) performed on the functional requirement by the ST author.

927 For example, the FDP_ACF.1 component contains four elements, each of which has complex assignments. If the ST contained, for instance, 10 rules in the FDP_ACF.1.2 assignment, and these 10 rules were implemented in specific places in 15 modules, it would be inadequate for the evaluators to "map" FDP_ACF.1.2 to the set of 15 modules and claim they had completed the work unit. Instead, the evaluators would map FDP_ACF.1.2 (rule 1) to modules A, B, and C; FDP_ACF.1.2 (rule 2) to modules A, C, and D; etc.

928 "user-visible" should be interpreted such that for requirements that have little or no manifestation at the user interface (e.g., FPT_SEP, FPT_RVM) it is not expected that the evaluator completely map those requirements to the TSFI. The analysis for those requirements will be performed in the analysis for the architectural design (ADV_ARC).

ADV_LLD.1-16 The evaluator *shall examine* the low-level design to determine that it is an accurate instantiation of the user-visible TOE security functional requirements.

929 The evaluator ensures that each security requirement listed in the TOE security functional requirements section of the ST has a corresponding design description in the low-level design that accurately details how the TSF meets that requirement. This requires that the evaluator identify a collection of modules that are responsible for implementing a given functional requirement, and then examine those modules to understand how the requirement is implemented. Finally, the evaluators would assess whether the requirement was accurately implemented.

930 As an example, if the ST requirements specified a role-based access control mechanism, the evaluators would first identify the modules that contribute to this mechanism's implementation. This could be done by in-depth knowledge or understanding of the low-level design or by work done in the previous work unit. Note that this trace is only to *identify* the modules, and is not the complete analysis.

931 The next step would be to understand what mechanism the modules implemented. For instance, if the low-level design described an implementation of access control based on UNIX-style protection bits, the low-level design would not be an accurate instantiation of those access control requirements present in the ST example used above.

932 "user-visible" should be interpreted such that for requirements that have little or no manifestation at the user interface (e.g., FPT_SEP, FPT_RVM) it is not expected that the evaluator completely map those requirements to the TSFI. The analysis for those requirements will be performed in the analysis for the architectural design (ADV_ARC).